EDILZA ROMANICHEN

UMA PROPOSTA DE APLICAÇÃO INTEGRADA PARA A TRANSFERÊNCIA AUTOMÁTICA DE DOCUMENTOS ESTRUTURADOS POR MEIO DA WEB, UTILIZANDO SERIALIZAÇÃO DE OBJETOS E META-LINGUAGEM XML

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, na Área de Engenharia de Software, Curso de Mestrado em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador:
Prof. Dr. Carlos Alberto Picanço de Carvalho

CURITIBA
2001

UFPR

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática da aluna ***Edilza Romanichen***, avaliamos o trabalho intitulado "***Uma Proposta de Aplicação Integrada para a Transferência Automática de Documentos Estruturados por Meio da Web, Utilizando Serialização de Objetos e Metalinguagem XML***", cuja defesa foi realizada no dia 19 de março de 2001. Após a avaliação, decidimos pela aprovação da candidata.

Curitiba, 19 de março de 2001.

Prof. Dr. Carlos Alberto Picanço de Carvalho
Presidente - Orientador

Prof. Dr. Dante Alves Medeiros Filho
Membro Externo - UEM

Prof. Dr. Laura Sanchez Garcia
DINF/UFPR

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

DEDICATÓRIA

Aos meus extraordinários pais,

Rosilda e Pedro;

e ao meu querido marido, companheiro desta caminhada,

Ewerton.

AGRADECIMENTOS

Meus sinceros agradecimentos e imenso respeito à Universidade Federal do Paraná e ao corpo docente e funcionários do Departamento de Informática, em especial ao professor orientador Carlos Alberto Picanço de Carvalho, e às professoras Olga Regina Pereira Bellon e Cristina Duarte Murta.

Aos professores da Banca Examinadora, Laura Sanchez Garcia e Dante Alves Medeiros Filho, por suas contribuições.

Aos professores do Setor de Ciências Sociais Aplicadas, Blênio César Severo Peixe e Wilson da Silva Spinosa, por sua recomendação e incentivo.

Ao Banco do Estado do Paraná S. A., pela oportunidade, e a todos os colegas de trabalho que testemunharam a conquista deste objetivo, pelo apoio e solidariedade, *valeu*!!!

Aos familiares, pela sempre portentosa compreensão, suporte e carinho: Cláudio, Patrícia e Bruno; Letícia e Fabiano; Aida Maris, Fabiano, Alexandre, Thiaguinho e Fabrício; Jefferson e Ivane; Emerson, Jandi, Emmanuel, Thiago, Lívia e Lucas; aos sogros Ayda e Cesário, e aos demais Romanichens, Oliveiras, Peres e Anjos!

Aos amigos, pela paciência por tantas ausências minhas: Alícia, Aninha, Aoqui, Castro, Cláudia e Onofre, Denílson e Cristina, Divonsir, Eliane, Fátima, Roseli, Sérgio e Lucimara, Simone, a todos os colegas de mestrado, Aldri, João Matias, Neilor, Rudá, e em especial, Luis Dias Pereira.

A Deus.

A VITÓRIA DA VIDA

Pobre de ti se pensa ser vencido
Tua derrota é caso decidido
Queres vencer, mas como, se em ti não crês
Tua descrença esmaga-te de vez
Se imaginares perder, perdido estás
Quem não confia em si marcha para traz
A força que te impele para frente
É a decisão firmada em tua mente

Muita empresa esboroa-se em fracasso
Ainda antes do primeiro passo
Muito covarde tem capitulado
Antes de haver a luta começado
Pensa em grande e teus feitos crescerão
Pensa em pequeno e irás depressa ao chão
O querer e o poder arquipotente
É a decisão firmada em tua mente

Fraco é aquele que fraco se imagina
Olha ao alto que ao alto se destina
A confiança em si mesmo é a trajetória
Que leva aos altos cimos da vitória
Nem sempre o que mais corre a meta alcança
Nem mais longe o mais forte o disco lança
Mas o que, certo em si, vai firme em frente
Com a decisão firmada em tua mente

Autor desconhecido.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ANSI	-	American National Standards Institute
API	-	Application Program Interface
ASCII	-	American National Standard Code for Information Interchange
CSS	-	Cascading Style Sheets
DOM	-	Document Object Model
DSSSL	-	Document Style, Semantic and Specification Language
DTD	-	Document Type Declaration
DTD	-	Document Type Definition
EBCDIC	-	Extended Binary Coded Decimal Interchange Code
EDI	-	Eletronic Data Interchange
E/S	-	Entrada e saída
HTML	-	HiperText Markup Language
HTTP	-	HiperText Transfer Protocol
IP	-	Internet Protocol
ISAPI	-	Internet Server API
JDK	-	Java Development Kit
JIT	-	Just-In-Time
JVM	-	Java Virtual Machine
NSAPI	-	Netscape Server API
OTP	-	Open Trading Protocol
PI	-	Processing Instructions
RTF	-	Rich Text Format
SGML	-	Standard Generalized Markup Language
TCP	-	Transmission Control Protocol
TEI	-	Text Encoding Initiative
TeX	-	Typesetting system
UCS	-	Universal Character Set
UML	-	Unified Modeling Language
URL	-	Uniform Resource Locator

URI	-	Uniform Resource Identifier
UTF	-	UCS Transformation Format
VCs	-	validity constraints
W3C	-	www consortium
WFCs	-	Well-formedness constraints
WYSIWYG	-	what you see is what you get
WWW	-	world wide web
XML	-	eXtensible Markup Language
XSL	-	eXtensible Style Language

RESUMO

A dificuldade de localizar e resgatar conteúdos na WEB, pela falta de padrões na estruturação de informações, pode gerar incompatibilidades na troca de dados entre pessoas e/ou organizações. Este fato motivou este trabalho, o qual propõe uma alternativa integrada de solução para a transferência automática e confiável de documentos estruturados no ambiente WEB, utilizando como recursos a serialização de objetos e a meta-linguagem XML. A alternativa de solução proposta se serve do mapeamento de atributos de objetos para o interior de um documento estruturado no formato XML, está baseada na arquitetura Internet, no modelo cliente-servidor e possui características como a garantia de entrega de dados e orientação à conexão. Esta proposta automatiza a solicitação de objetos serializados por meio da WEB feita pela máquina cliente, produzida pelo emprego de um programa que usa soquete. Também automatiza a resposta processada pelo servidor WEB, servindo-se de um processo que gera uma resposta dinâmica. Uma aplicação foi desenvolvida para validar os requisitos da proposta apresentada neste texto. Um experimento utilizando esta aplicação foi conduzido com o objetivo de observar a aplicabilidade desta proposta, comprovando o funcionamento das partes e comunicabilidade integral.

Palavras-chave: Serialização de Objetos; XML; WEB; Internet.

ABSTRACT

The difficulty of locating and retrieving contents from the WEB, due to the absence of standards in the structuring of information, can generate incompatibilities in data interchange among people and / or organizations. This fact has motivated this work. It proposes an alternative of an integrated solution for the automatic and reliable transfer of structured documents in the WEB environment, using as a resource the serialization of objects and the meta-language XML. The alternative of the proposed solution, which uses the mapping of attributes of objects into the interior of a document structured in the format XML, is based on the Internet architecture and on the customer-server model. It also presents characteristics such as data delivery warranty and orientation to the connection. This proposal automates the solicitation of serialized objects, by means of the WEB, done by the machine customer and generated through the utilization of a program that uses socket. It also automates the answer processed by the WEB server through the utilization of a process that generates a dynamic answer. An application has been developed to validate the requirements of the proposal presented in this text. Also an experiment using this application has been performed. The main objective of this experiment has been the observation of the applicability of the proposal and checking the operation of the parts and integral communication.

Key-words: Object Serialization, XML, WEB, Internet.

1. INTRODUÇÃO

1.1 MOTIVAÇÃO

A *Word Wide Web* (WWW), ou simplesmente WEB, é um modelo de uma grande biblioteca digital, distribuída e ao alcance de milhares de máquinas ao redor do mundo pelo uso da Internet (LESK, 1997, p. 21-25).

A utilização da WEB em redes de alta velocidade, como a Internet2 (IBM, 1998, p. 462-466), somada ao uso de novos protocolos de comunicação, como o *Internet Protocol Next Generation* (IPng), que tornam disponíveis um número maior de endereços de máquinas conectadas à Internet (FORJAN, 2001) (IBM, 1998, p. 357-400), apresenta um meio favorável e oportuno ao desenvolvimento de aplicações. Entre estas, encontram-se as aplicações WEB. De acordo com MARUYAMA, TANAMURA e URAMOTO (1999, p. 6-7) uma aplicação WEB é qualquer aplicação ou sistemas de aplicações que utilizem como seu protocolo de transporte primário o *HiperText Transfer Protocol* (HTTP) (IBM, 1998, 440-448) (WON, 2000) e são geralmente construídas seguindo o modelo de três camadas: camada de interface gráfica, camada de aplicação ou negócio e camada de banco de dados. A camada de interface gráfica é executada na máquina cliente WEB, a camada de negócio é processada na máquina servidora WEB e a camada de banco de dados é executada na máquina servidora de banco de dados (MSDN, 1999a).

O acesso a documentos na WEB é feito através da utilização de programas navegadores que processam em máquinas clientes WEB (IBM, 1998, p. 437-439). A obtenção da informação requerida necessita de um tratamento prévio de interpretação pelo cliente, bem como da extração dos elementos desejados (LESK, 1997, p. 25-26).

Pela falta de padrões nas trocas de informações entre pessoas e/ou organizações, as diferentes formas de estruturação da informação na WEB causam problemas tais como a incompatibilidade entre os geradores e os receptores de documentos (SANTOS, RESENDE, 2000).

RAMALHO (2000) considera que um documento é criado com o objetivo de registrar a informação e para que tenha valor, é necessário que seja fácil de

localizar, interpretar, validar e reutilizar. Cita também que a estrutura de um documento é a chave para o acesso de suas informações e um meio de tornar explícita esta estrutura é adicionar ao seu conteúdo uma anotação extra ou marca textual, para identificar elementos de conteúdo, estrutura e apresentação de um texto, como por exemplo o cabeçalho, a data e o autor, entre outros. Documentos que usam estas marcas textuais são conhecidos por documentos estruturados.

A evolução de tecnologias de tratamento de documentos com a adição de marcas textuais que definem a estrutura de seus conteúdos, durante as últimas quatro décadas, resultaram na recomendação da meta-linguagem XML como um padrão de mercado (BRAY; PAOLI; SPERBERG-McQUEEN, 1998).

As coletas rotineiras de documentos e informações em páginas conhecidas são, muitas vezes, realizadas por usuários através de procedimentos manuais. Em grandes volumes, este trabalho pode se tornar impraticável. Nota-se que a presença do usuário é necessária para atividades como solicitação de páginas WEB, interpretação e extração de conteúdo e gravação de arquivos na máquina cliente. Por questões de segurança, serviços de gravação de arquivos devem ser solicitados pelo cliente (LESK, 1997, p. 25-26).

Dentro do paradigma de orientação a objetos, percebe-se que os documentos textuais, sons, vídeos e informações disponíveis na WEB podem ser entendidos como objetos, pois segundo RUMBAUGH et al. (1994, p. 31), "Um objeto é um conceito, uma abstração, algo com limites nítidos e significados em relação ao problema em causa". A transformação de atributos de objetos em uma seqüência de bytes recebe o nome de "serialização" (JOHNSON, 199?).

Desta forma, pode-se ter objetos sendo publicados em páginas nos servidores WEB, podendo ser coletados em forma de mensagens por usuários em máquinas clientes WEB.

Nota-se a necessidade de se desenvolver um estudo sobre a transferência de objetos por meio da WEB, baseado em sua serialização para o formato XML, que se propõe a resolver o problema de incompatibilidade entre os padrões de estruturação de documentos. A identificação de alternativas tecnológicas que facilitem as aplicações de negócios eletrônicos neste meio e a automatização dos

procedimentos manuais rotineiros de coletas de documentos e informações pelos usuários também são desejáveis.

Para que processos computacionais possam se beneficiar desta automatização, interpretando e extraindo as informações requeridas, uma aplicação WEB terá maior utilidade se compuser uma solução integrada que envolva, de um lado, a responsabilidade pela publicação de objetos como documentos estruturados por parte dos servidores WEB, e de outro, a habilidade de tornar disponível uma interface na máquina cliente WEB capaz de automatizar os procedimentos de solicitação de páginas e de gravação de arquivos contendo os objetos documentos estruturados resultantes desta solicitação.

1.2 OBJETIVOS

Este trabalho propõe uma aplicação WEB integrada como alternativa para solucionar o problema de coleta de conteúdo textual publicado na WEB, através de serialização de objetos utilizando o padrão XML. Esta aplicação WEB integrada será definida com base na execução de processos necessários na máquina cliente e no servidor WEB, sendo que um experimento será conduzido com a finalidade de observar a adaptabilidade da aplicação proposta.

Para tanto, alguns objetivos específicos são destacados:

a) identificar pelo menos uma alternativa tecnológica para compor uma solução integrada de transferência de objetos serializados;
b) caracterizar uma alternativa de serialização de objetos inserida no meio WEB e processos de distribuição e persistência associados;
c) integrar tecnologias independentes de fabricantes e linguagens;
d) validar a solução proposta através de um experimento que demonstre a sua aplicabilidade.

Algumas restrições na condução deste trabalho foram estabelecidas, como:

a) não criar um novo protocolo, utilizando somente protocolos disponíveis;
b) não tratar aspectos de segurança.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

Os conteúdos dos capítulos que compõem a estrutura desta dissertação são descritos na seqüência.

Capítulo 2, Revisão Bibliográfica e Conceitos Básicos. Apresenta características, funcionalidades básicas, define conceitos, padrões e terminologias associadas à meta-linguagem XML. Explora os conceitos de distribuição e persistência associados a serialização, que são a base para a proposta da aplicação integrada, e caracteriza o ambiente de aplicações WEB.

Capítulo 3, Proposta de Aplicação WEB Integrada. Contém a descrição da proposta de uma aplicação para realizar a transferência de documentos estruturados por meio da WEB, baseada em serialização de objetos, utilizando a meta-linguagem XML.

Capítulo 4, Experimento. São apresentados a introdução, metodologia, recursos utilizados na implementação, os resultados obtidos e algumas considerações a respeito do experimento conduzido neste trabalho.

Capítulo 5, Conclusão. Este capítulo resume as contribuições surgidas a partir desta dissertação, encerrando o entendimento gerado a partir da execução deste trabalho.

2. REVISÃO BIBLIOGRÁFICA E CONCEITOS BÁSICOS

Neste capítulo são citados os trabalhos existentes na literatura, relacionados à transferência eletrônica de documentos e meta-linguagem XML, sendo apresentados os conceitos básicos e terminologias associadas que fazem parte dos padrões recomendados e que são utilizados nesta dissertação. São fornecidos os conceitos de objeto e serialização de objetos somados aos de persistência e distribuição. É também apresentado o ambiente de aplicações WEB.

2.1 META-LINGUAGEM XML

Esta seção apresenta um histórico evolutivo do tratamento de documentos, o documento XML e conceitos básicos associados.

2.1.1 Evolução do Tratamento de Documentos

A troca e a manipulação de documentos estruturados publicados utilizando um padrão aberto são ambições desde a década de 60. Nesta época um comitê da *Graphic Communications Association (GCA)* criou a linguagem GenCode, que permitiu aos seus clientes enviarem os mesmos dados de diversos modos para diferentes vendedores. Em paralelo, a IBM desenvolveu a linguagem *Generalized Markup Language* (GML), para que as suas publicações internas como livros, relatórios e edições eletrônicas pudessem ser produzidos a partir do mesmo arquivo fonte (KHARE, 1997a, p. 78-80).

No início dos anos 80 representantes das comunidades responsáveis pelas linguagens GenCode e GML reuniram-se com o objetivo de padronizar a especificação, definição e utilização de marcas textuais em documentos, e, em 1986, o comitê *American National Standards Institute* (ANSI) de Linguagens de Computação para o Processamento de Texto publicou a meta-linguagem *Standard Generalized Markup Language* (SGML) como o padrão ISO 8879:1986, sendo que seus dois elementos chave foram a sintaxe da linguagem GML e a semântica da linguagem *GenCode* (KHARE, 1997a, p. 80).

Em paralelo a estes projetos de criação e padronização de marcas textuais em documentos, surge entre os anos 70 e 80 a rede hoje chamada de "Internet", a qual é caracterizada pela estrutura física que permite transferência de mensagens (BEN-NATAN, 1997).

A *Word Wide Web* – WWW, ou WEB, baseada na Internet, é caracterizada por conter documentos, sons, vídeos e informações distribuídas e ao alcance de milhares de máquinas ao redor do mundo. A WEB teve início em 1989 no *Conseil Européen pour la Researche Nucleaire* (CERN), localizado em Genebra, através de um projeto proposto por Tim Berners-Lee sobre um sistema de informação descentralizado que se apoiava na Internet, permitindo ligar informações relacionadas entre si. Neste mesmo ano, Marc Andreesen da Universidade de Illinois, mais tarde fundador da empresa Netscape, lançou a primeira versão do programa Mosaic, que permitiu a navegação pela rede com uma interface gráfica (LESK, 1997, p. 21) (KHARE, RIFKIN, 1997, p. 80).

Em 1990, Tim Berners-Lee criou a linguagem *HiperText Markup Language* (HTML) a partir da escolha de um subconjunto de conceitos da linguagem SGML, para que seu programa navegador pudesse entender marcações textuais sintáticas dentro de documentos (LESK, 1997, p. 80) (KHARE, RIFKIN, 1997, p. 80). O desenvolvimento destes projetos tornou disponível a utilização de marcas textuais em documentos dentro da linguagem HTML reconhecida por programas de navegação na WEB.

Em 1994 se formou o consórcio *Word Wide Web Consortium* (W3C), cujo objetivo é desenvolver especificações de protocolos comuns para a WEB. Em julho de 1996 iniciou-se um grupo de estudo sobre a meta-linguagem SGML, o qual trabalhou para incorporar os benefícios de sintaxe e semântica desta meta-linguagem para a WEB, conservando um formato utilizável por pessoas e por computadores. Este trabalho resultou na especificação da meta-linguagem *eXtensible Markup Language* (XML), a qual foi recomendada em 1998 como um padrão pelo consórcio W3C (KHARE, RIFKIN, 1997, p. 80).

XML é uma meta-linguagem. Assim, admite a criação de linguagens de marcação textual, permitindo que o autor determine seu próprio conjunto de marcas textuais para diferentes classes de documentos. Esta meta-linguagem é um padrão,

não sendo propriedade de nenhuma empresa. Significa que grupos de pessoas ou organizações poderão definir uma linguagem para trocar informações em seus domínios de atividade, como por exemplo música, química, matemática e comércio, entre outros (USDIN, GRAHAM, 1998, p. 125-127).

O núcleo da meta-linguagem XML é o resultado da análise do significado do termo "documento" no mundo digital, que consiste da somatória de três elementos distintos: conteúdo, estrutura e apresentação, e na idéia de que manter estes componentes separados trará benefícios significativos para a automatização de processos computacionais. Estes documentos são chamados "documentos XML" (McGRATH, 1998, p. 16-17).

Documentos XML válidos são documentos válidos e reconhecidos pela SGML. A meta-linguagem XML é baseada em padrões internacionais de apresentação (DSSSL - *Document Style, Semantic and Specification Language* - ISO/IEC 10179 mais CSS - *Cascading Style Sheets*), de captura de informações em hipertexto (HyTime ISO/IEC 10744 mais TEI - *Text Encoding Initiative*) e seu conjunto de caracteres nativos é o UNICODE (ISO/IEC 10646) (McGRATH, 1998, p. 25).

As informações disponíveis na WEB escritas em HTML dificultam a ação de agentes de software e mecanismos de buscas por não se encontrarem codificadas semanticamente. A codificação da informação dentro de documentos estruturados e padronizados semanticamente, como é o caso de documentos XML, promete a abertura do comércio eletrônico na medida em que propicia a automação de processos computacionais e permite a troca de documentos comerciais, facilitando o acesso da WEB por programas agentes, que poderão pesquisar, negociar e efetuar compras (GLUSHKO; TENEMBAUN; MELTZER, 1999, p. 106).

Várias são as entidades que buscam uma padronização de especificações baseadas na meta-linguagem XML para o domínio de negócios, o que capacitaria a interoperabilidade e simplificaria o compartilhamento e reutilização de informações. Alguns destes exemplos são o *Open Trading Protocol* (OTP), que especifica um padrão para necessidades de informações para pagamentos, recibos, entrega e suporte ao consumidor; XML/EDI (*Eletronic Data Interchange*), que define como seus elementos podem ser representados utilizando a meta-linguagem XML;

RosettaNet, que trata a troca de catálogos de produtos e transações entre fabricantes, distribuidores e revendedores (GLUSHKO; TENEMBAUN; MELTZER, 1999, p. 109).

2.1.2 Documento Estruturado

O termo "documento" no mundo digital consiste do uso equilibrado de três elementos distintos: conteúdo, estrutura e apresentação.

A essência do paradigma chamado *What You See Is What You Get* (WYSIWYG), que significa "o que você vê é o que você tem", é tornar transparente para o usuário a integração entre o conteúdo e a apresentação de um documento. Nos editores de texto desta natureza, por exemplo, criam-se documentos com a preocupação voltada para a aparência de uma apresentação em particular, como um dispositivo de saída específico. As únicas informações armazenadas sobre a estrutura são detalhes sobre margens e fontes, entre outras, relacionadas à publicação do documento (McGRATH, 1998, p. 16-17).

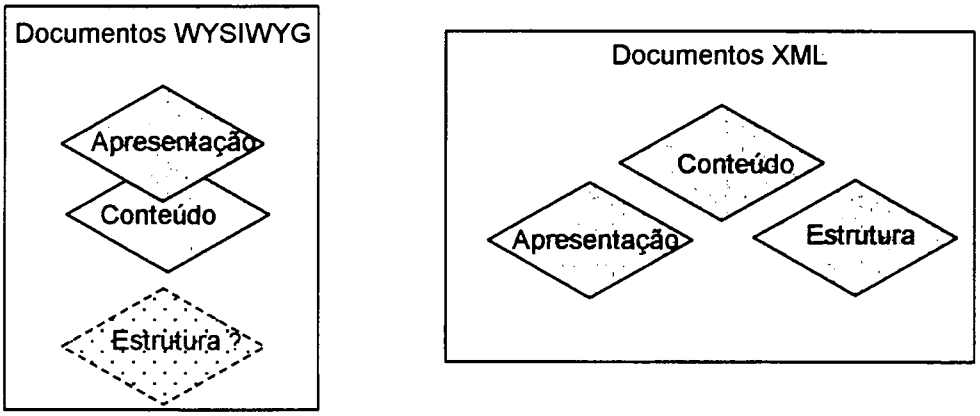
Em contrapartida, a idéia central da filosofia XML é manter os elementos estrutura, conteúdo e apresentação separados de forma explícita, para que processos computacionais possam obter ganhos significativos a partir das declarações de estruturas. Considera-se a estrutura inerente ao documento tão importante quanto o seu conteúdo, sendo que a apresentação, também considerada importante, pode ser mantida separadamente. Para tal, o documento necessita ser expresso de maneira a expor seu tipo, a organização entre seus dados, quais tipos de dados contém e em que ordem os mesmos aparecem. (McGRATH, 1998, p. 16-17). USDIN e GRAHAM (1998, p. 126) endossam esta visão, ressaltando que os princípios fundamentais da filosofia XML são a separação do conteúdo do formato, a hierarquia das estruturas de dados, a marcação embutida e as estruturas definíveis pelo usuário.

A figura 1 representa graficamente um exemplo de visões de documentos WYSIWYG e de documentos XML.

A meta-linguagem XML descreve uma classe de objetos de dados chamados de documentos XML e possui uma família de ferramentas associadas, ou

módulos opcionais. Em (W3C, 2000) podem ser obtidas informações sobre estes módulos.

FIGURA 1 – DOCUMENTOS WYSIWYG E XML



FONTE: McGRATH, S.. **XML by Example – Building E-Commerce Applications.** New Jersey : Prentice-Hall, p:17, 1998.

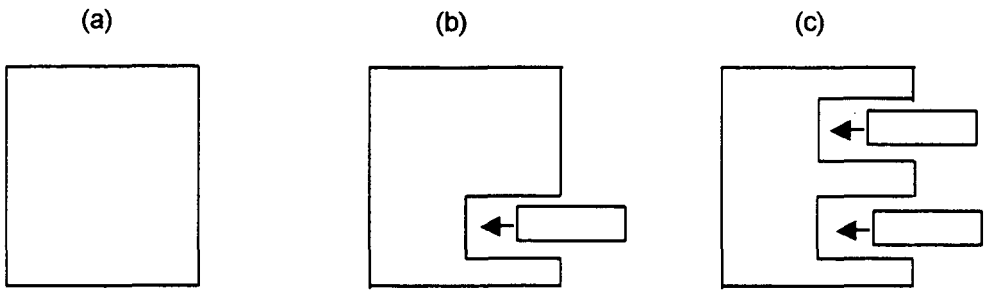
Para um documento estar em conformidade com o padrão XML ele deve obrigatoriamente estar de acordo com a especificação XML 1.0 (BRAY et al., 2000).

2.1.2.1 Estrutura de um documento XML

A estrutura de um documento XML pode ser vista de duas maneiras: física e logicamente. A estrutura física é composta por unidades de armazenamento, ou arquivos físicos, chamadas "entidades". A unidade raiz do documento XML é chamada de "entidade documento". Um documento XML pode consistir de uma ou mais unidades de armazenamento. Uma entidade pode fazer referência a outra entidade, ocasionando sua inclusão no documento. A figura 2 mostra três diferentes documentos XML e suas entidades.

A estrutura lógica de um documento XML é composta por declarações, elementos, comentários e instruções de processamento, entre outros, marcados fisicamente no documento. De uma forma geral, um documento XML se apresenta dividido em prólogo (ou cabeçalho), elemento raiz e rodapé (ou epílogo), como a figura 3 apresenta.

FIGURA 2 – DOCUMENTOS XML

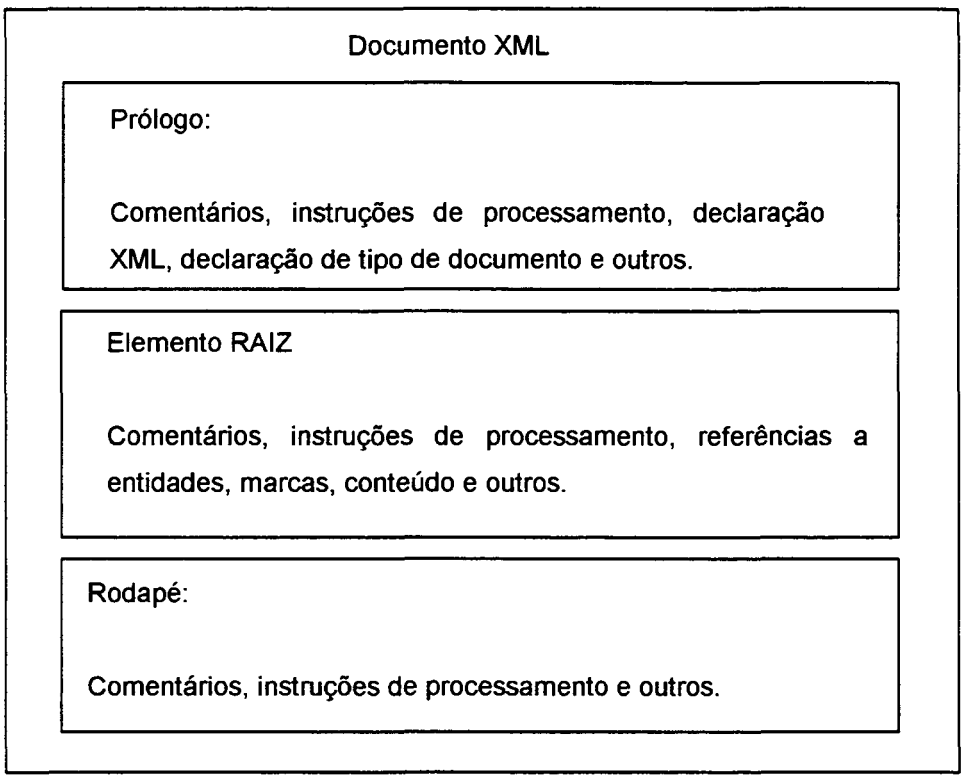


(a) somente uma "entidade documento";

(b) uma entidade documento que faz referência a outra "entidade";

(c) uma entidade documento que faz referência a outras duas "entidades".

FIGURA 3 – DIVISÃO DE DOCUMENTO XML



O prólogo pode apresentar a declaração XML, comentários, instrução de processamento e declaração de tipo de documentos, entre outros. Os comentários são opcionais e delimitados entre "<!--" e "-->". As instruções de processamento são opcionais, abreviadas como PI (*Processing Instructions*) e utilizadas para inserir instruções não-XML no documento. A declaração de tipo de documento é opcional.

O elemento raiz, ou elemento documento, indica a estrutura lógica do documento associada a seu conteúdo. Pode conter comentários, instruções de processamento, marcas textuais, referências a outras entidades, referências a outros documentos adicionais, entre outros. O rodapé é opcional e pode apresentar comentários e instruções de processamento.

2.1.2.2 Conteúdo do documento XML

O documento XML é composto por uma ou mais unidades de armazenamento que podem conter dados XML e não XML. Dados XML são sempre caracteres e contêm marcas e conteúdo textuais. Dados não XML podem ser gráficos, vídeos, arquivos, entre outros.

As entidades de um documento XML que possuem dados no formato XML têm uma combinação de marcações textuais e dados. As marcações textuais incluem marcas padrão iniciais e finais e comentários, entre outras. Estas marcas padrão são palavras contidas entre os sinais tipográficos "<" (menor) e ">" (maior) e estão embutidas nos documentos XML para identificar onde cada elemento inicia e acaba. A figura 4 mostra um exemplo de um documento XML, tendo na primeira linha a declaração XML e na segunda o elemento raiz, **Extrato**. As marcas estão representadas em negrito, e os conteúdos estão localizados entre as marcas textuais.

FIGURA 4 – EXEMPLO DE UM DOCUMENTO XML

```
<?xml version="1.0" ?>
<Extrato>
    <Cliente>Asdrúbal Oliveto</Cliente>
    <Operacao>
        <NumBanco>25</NumBanco>
        <NumAgencia>62</NumAgencia>
        <NumConta>12345</NumConta>
        <SldInic>1200,00</SldInic>
    </Operacao>
</Extrato>
```

Em XML entidades podem estar baseadas em diferentes codificações de caracteres, como por exemplo, UTF-8 (*UCS Transformation Format,* onde UCS significa *Universal Character Set*) (ROBERTS; HELLER; ERNEST, 2000, p. 438-439), UTF-16, ASCII (*American National Standard Code for Information Interchange*), EBCDIC (*Extended Binary Coded Decimal Interchange Code*) entre outros. Um processador XML pode, a partir dos primeiros bytes de um documento, detectar a codificação utilizada, como o exemplo mostrado pelo quadro 1.

QUADRO 1 - EXEMPLOS DE CODIFICAÇÕES DE UM DOCUMENTO XML

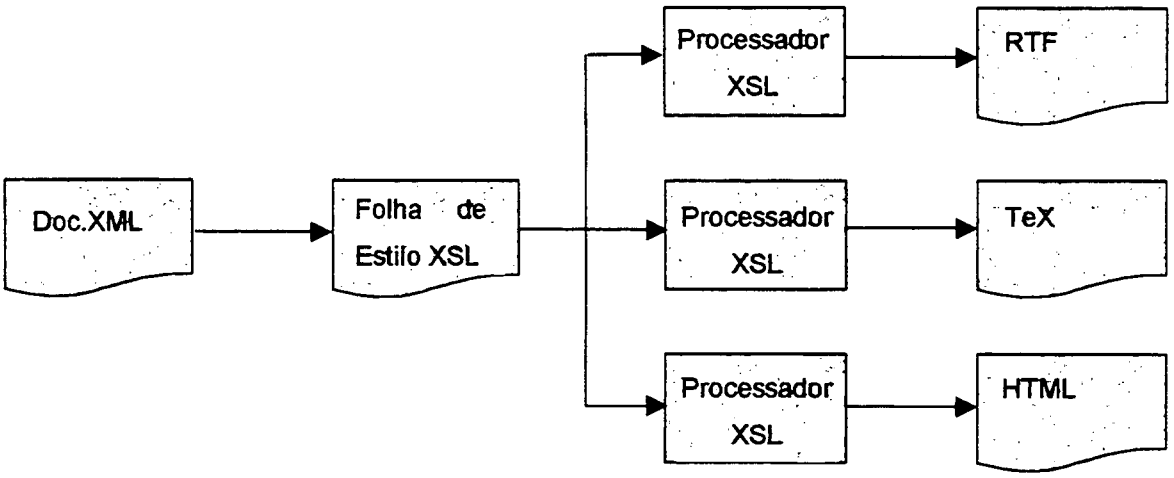
Primeiros bytes de uma entidade XML (em Hexadecimal)	Conclusões de um processador XML
FE FF	UTF-16
FF FE	UTF-16
3C 3F 78 6D	UTF-8, ASCII, ISO-646, e outros.
4C 6F A7 9A	EBCDIC

2.1.2.3 Apresentação do documento XML

A apresentação de um documento XML pode estar contida em outra unidade chamada "folha de estilo XSL" (Linguagem de Estilo XML – *eXtensible Style Language*), que é opcional. O conjunto de regras conhecidas como "regras de construção", especificadas dentro de uma folha de estilo XSL, definem como o documento fonte XML será convertido dentro de uma hierarquia de objetos formatados e diz ao processador como converter um documento XML em outra notação, como por exemplo, HTML, *Typesetting System* (TeX) e *Rich Text Format* (RTF), entre outros. Esta hierarquia é conhecida como "objetos de fluxo" e inclui conceitos de página, parágrafo e célula de tabela, por exemplo (McGRATH, 1998, p. 309-334).

A figura 5 apresenta as estruturas envolvidas na conversão de documentos XML para outros formatos através da utilização de folhas de estilo XSL:

FIGURA 5 - CONVERSÃO DE DOCUMENTO XML UTILIZANDO FOLHA DE ESTILO XSL



FONTE: McGRATH, S.. **XML by Example – Building E-Commerce Applications.** New Jersey : Prentice-Hall, p:310, 1998.

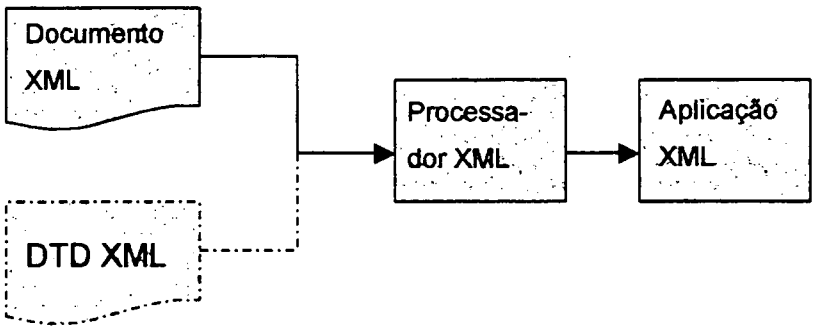
2.1.3 Parsers XML

O *parser*, que é o programa processador, é o responsável pela interpretação das marcas e estruturas de dados de documentos XML, servindo de ligação com a aplicação (McGRATH, 1998, p. 22, 114). Esta análise é denominada verificação ou *parsing*.

Existem duas classes de processadores XML que verificam a validade e formação de documentos XML: validadores e não-validadores. Os processadores validadores verificam se o documento XML está de acordo com a definição DTD referenciada em seu cabeçalho, ou seja, se é válido para esta classe. Os não-validadores verificam se o documento é bem-formado.

A figura 6 representa as entradas e saída do processo de *Parsing*, onde a definição DTD é opcional.

FIGURA 6 - ENTRADAS E SAÍDA DO PROCESSO DE PARSING



FONTE: McGRATH, S.. **XML by Example – Building E-Commerce Applications.** New Jersey : Prentice-Hall, p:114, 1998.

2.1.4 Classes de Documentos XML

Para um documento estar em conformidade com o padrão XML deve obrigatoriamente ser "bem-formado", estando de acordo com a Recomendação XML 1.0. O termo bem-formado significa que o documento segue as restrições *Well-Formedness Constraints* (WFCs) definidas pela Recomendação XML 1.0, as quais estabelecem limites em relação à estrutura física dos elementos. Significa que as unidades XML do documento XML são bem-formadas. Em outras palavras, unidades XML possuem, por exemplo, as marcas iniciais e finais no devido lugar e aninhadas corretamente.

Um documento XML bem-formado também pode ser considerado válido segundo a sua interpretação. O termo válido, dentro do contexto XML, significa que um documento bem-formado deve seguir também as restrições *Validity Constraints* (VCs) especificadas na Recomendação XML 1.0, as quais estipulam limites em relação à estrutura lógica dos elementos. Para um documento XML poder ser validado por um *parser*, deve possuir obrigatoriamente uma definição DTD associada, sendo que todos os elementos e atributos do documento devem estar de acordo com a definição DTD. Todo o documento válido é também bem formado, porém o inverso não é verdadeiro (MARUYAMA; TANAMURA; URAMOTO, 1999, p.15-18).

2.1.5 Definição de Tipo de Documento - DTD

A definição DTD, não obrigatória para se considerar um documento bem-formado, estabelece um conjunto de regras que definem o formato de documentos válidos em XML. Especifica um conjunto de elementos obrigatórios e opcionais e seus atributos para documentos em conformidade com este tipo, bem como os nomes de marcações textuais e o relacionamento entre os elementos dentro do documento (KHARE, RIFKIN, 1997a). Estas regras são passíveis de checagem por meio do *parser*.

A Declaração de Tipo de Documento faz referência a uma definição DTD associada ao documento XML e deve estar localizada no cabeçalho do documento XML. Pode apontar para uma unidade de armazenamento que a contenha ou efetivamente conter a definição DTD.

2.1.6 Modelo de Objeto Documento - DOM

O *Document Object Model* (DOM) é uma definição de interface abstrata baseada em uma estrutura de árvore recomendada como padrão em outubro de 1998 pelo consórcio W3C (APPARAO et al., 1998).

A especificação DOM Nível Um permite que programas e *scripts* acessem e atualizem automaticamente o conteúdo, a estrutura e o estilo de documentos XML. A interface DOM define um conjunto padrão de objetos para representar documentos XML e HTML, um modelo padrão para operações através desses objetos e uma interface padrão para manipulá-los.

Programas que implementam estas definições tornam disponível o acesso às marcas e às estruturas de dados de documentos XML, permitindo a leitura e a alteração de dados pela aplicação. Este modelo pode ser utilizado para acessar o conteúdo de um documento XML.

2.1.7 SGML x XML

Um dos objetivos do projeto da meta-linguagem XML foi mantê-la compatível com a especificação SGML, que é uma meta-linguagem robusta, poderosa e complexa, orientada a organizações com dezenas de milhares de páginas de informações. A meta-linguagem XML tem a intenção de tornar a abordagem da meta-linguagem SGML utilizável dentro do ambiente WEB, sendo um subconjunto simplificado da meta-linguagem SGML. Isto significa que todos os documentos XML são documentos válidos para a meta-linguagem SGML, porém o inverso não é verdadeiro (McGRATH, 1998, p. 20-21).

2.2 SERIALIZAÇÃO DE OBJETOS, PERSISTÊNCIA E DISTRIBUIÇÃO

Segundo definição de RUMBAUGH et al. em (1994, p. 31), "Todos os objetos têm identidade e são distinguíveis. Uma classe de objetos descreve um grupo de objetos com atributos, operações e semântica comuns. Um atributo é uma propriedade dos objetos de uma classe; uma operação é uma ação que pode ser aplicada aos objetos de uma classe.".

Quando um objeto está ativo na memória, estruturas complexas de dados são transformadas em tipos de dados simples que durante a serialização devem ser perfilados lado a lado, transformando-se em uma cadeia de bytes de apenas uma dimensão. O processo de serialização não atua sobre as operações de uma classe mas sobre seus atributos. As operações do objeto podem ser exportadas em um arquivo separado na linguagem utilizada caso solicitado.

A serialização associa conceitos de persistência e distribuição. Persistência pode ser definida como a propriedade em que um objeto continua a existir apesar do fim de sua execução. Esta existência pode-se dar em relação ao tempo ou localização. A distribuição de objetos é a transmissão da persistência para outro local (JOHNSON, 199?) (MSDN, 1999b) (PIANESSO, CASTANHO, 2000) (BANKSTON, SEIFERT, 1997).

2.3 AMBIENTE DE APLICAÇÕES WEB

A WEB é baseada na rede Internet, que por sua vez utiliza o conjunto de protocolos TCP/IP. Muitas aplicações Internet TCP/IP adotam o modelo cliente-servidor para transferência de dados. Nas próximas sessões são identificados os componentes da arquitetura Internet e são apresentadas as camadas da aplicação WEB

2.3.1 Protocolos de Transporte da Arquitetura Internet TCP/IP

O Protocolo de Transporte *Transmission Control Protocol* (TCP) (SOARES; LEMOS; COLCHER, 1997, p. 311-329) é um dos protocolos da camada de transporte do conjunto TCP/IP utilizado na arquitetura Internet. Fornece um serviço de transferência de cadeias de dados confiável e opera no modo orientado à conexão (circuito virtual).

O protocolo de transporte TCP interage de um lado com processos da camada de aplicação e do outro com o protocolo *Internet Protocol* (IP) da camada de rede, e se responsabiliza pela recuperação de dados corrompidos, perdidos, duplicados ou entregues fora de ordem pelo protocolo da camada de rede. O conjunto de protocolos TCP/IP utiliza o conceito abstrato de porta, associando-as a processos de aplicação, para permitir que vários processos possam simultaneamente usar serviços, como o protocolo de transporte TCP, em uma única máquina (máquina *host*).

2.3.2 O Nível de Aplicação Internet TCP/IP

Na arquitetura Internet TCP/IP os serviços da camada de aplicação (SOARES; LEMOS; COLCHER, 1997, p. 353-395) (IBM, 2000, p. 149-262) são implementados de forma isolada e trocam dados utilizando diretamente a camada de transporte através de rotinas genéricas, chamadas de soquetes. Alguns dos protocolos no nível da camada de aplicações disponíveis nesta arquitetura que usam o protocolo de transporte TCP são o *File Transfer Protocol* (FTP) e o HTTP .

O Protocolo FTP é específico para transferência de arquivos e permite que sejam transferidos arquivos do tipo texto (manipulados como cadeias de caracteres ASCII ou EBCDIC) e arquivos binários.

O protocolo HTTP foi projetado para permitir a transferência de documentos de hipertexto entre aplicações WEB e pode conter outros elementos à parte do texto, tais como imagens gráficas, clipes de áudio e vídeo, programa *applet* (programa que processa na máquina cliente), entre outros. Este protocolo é baseado no processo de requisição-resposta, sendo esta transação dividida basicamente em quatro passos (IBM, 2000, p. 440-448):

a) o programa cliente, podendo ser um navegador ou um aplicativo usando soquetes, abre uma conexão;

b) o programa cliente envia uma requisição ao servidor WEB;

c) o servidor envia uma resposta ao programa solicitante;

d) encerramento da conexão.

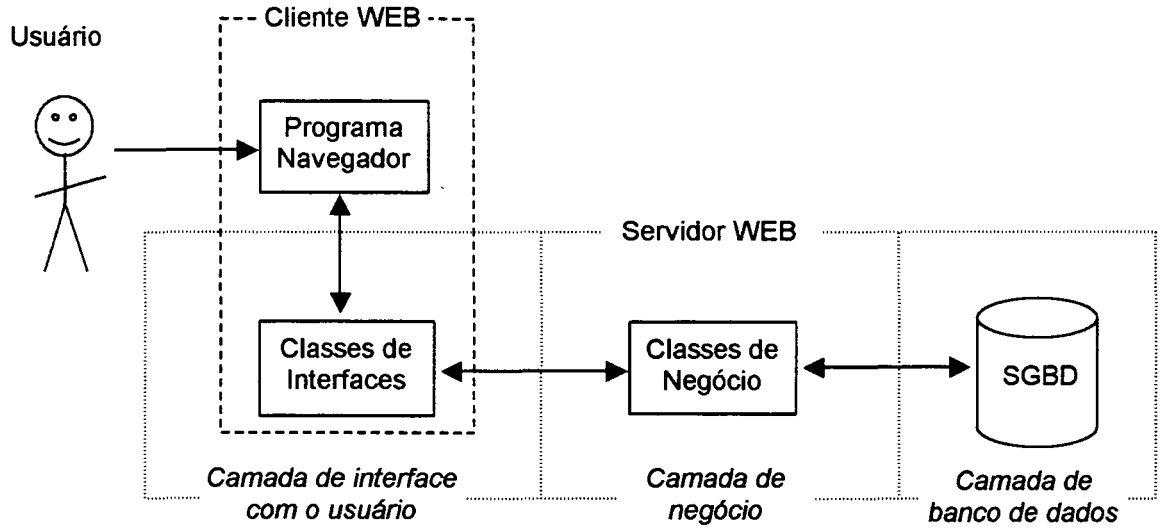
O protocolo HTTP, em sua versão mais recente, utiliza conexão persistente, mantendo a conexão de rede aberta para as múltiplas transações entre cliente e servidor (WONG, 2000).

Uma requisição HTTP pode também ser originada por um outro programa, que não o navegador, usando o mesmo modelo e protocolo utilizado pelo usuário humano, por meio de soquetes. Este programa pode resolver a limitação do programa navegador, imposta por questões de segurança, que não permite a gravação automática de arquivos na máquina cliente, com exceção de "*cookies*" de no máximo de 4Kb, quando habilitados nas opções de segurança do programa navegador.

2.3.4 Aplicação WEB e Arquitetura de Três Camadas

Uma aplicação WEB é geralmente construída com base numa arquitetura de três camadas, que a modulariza em camada de interface gráfica, camada de negócio e camada de banco de dados (MARUYAMA; TANAMURA; URAMOTO, 1999, p. 8), como mostrado na figura 7.

FIGURA 7 - ARQUITETURA DE TRÊS CAMADAS WEB



A interface gráfica utiliza um programa navegador, é composta por classes de interface da aplicação e é processada na máquina do usuário. A camada de negócio possui as classes de negócios, que são executados no servidor WEB e processam os dados do negócio. Na camada de banco de dados o sistema de gerenciamento de banco de dados, ou classes de persistência, armazena e/ou recupera os dados requisitados pelas outras duas camadas.

2.4 CONSIDERAÇÕES FINAIS

Este capítulo citou os trabalhos relacionados a documentos estruturados e forneceu uma visão sucinta de definições e recursos associados ao padrão XML. Foram apresentados conceitos de serialização, distribuição e persistência de objetos, bem como a caracterização do ambiente de aplicações WEB. Estes conceitos serão utilizados nos próximos capítulos, onde serão apresentadas as propostas de integração destes itens à plataforma de três camadas de aplicações WEB e um experimento realizado com uma pequena aplicação desenvolvida para observar a aplicabilidade desta proposta de integração.

3 PROPOSTA DE APLICAÇÃO WEB INTEGRADA

Neste capítulo são identificados os procedimentos necessários para automatizar as atividades de um cliente WEB em busca de conteúdo textual estruturado, sendo identificada a importância da inclusão de estrutura no processo de serialização de objetos. Associando os conceitos de serialização, persistência e distribuição de objetos ao ambiente de aplicações WEB são determinados os requisitos necessários para a proposta. Cada requisito destes é então aprofundado e são apresentadas alternativas para a utilização de métodos e recursos de tecnologia. Uma alternativa de solução integrada é então proposta com base nos itens anteriores.

3.1 CONTEXTUALIZAÇÃO DA APLICAÇÃO WEB INTEGRADA

Na abordagem de serialização, a separação e reconhecimento de itens de dados e também a preservação da disposição hierárquica de estruturas de dados aninhadas, como por exemplo, vetores multidimensionais, são elementos mantidos de forma posicional em um vetor de uma única dimensão. Esta característica faz com que implementações de serialização e seu processo inverso sejam interdependentes.

Para permitir a serialização de objetos contendo sua estrutura, a serialização deve admitir a inclusão de marcas textuais a este vetor unidimensional. Desta forma, a cadeia serializada guarda informações sobre nomes de itens de dados e armazena a estrutura hierárquica do objeto instanciado, de forma independente da sua posição inicial. Propomos a utilização deste recurso como um dos processos necessários à criação da aplicação integrada e sugerimos a utilização da meta-linguagem XML e a aplicação de suas definições para que as publicações de documentos resultantes possam ser a entrada de processos computacionais escritos em outras linguagens e também possam ser tratados pelos mecanismos fornecidos pelo padrão XML, como definições DOM e *parsers*, entre outras.

Associados ao processo de serialização definimos outros dois processos necessários à criação da aplicação integrada: o processo de distribuição, que significa a transferência dos objetos documentos estruturados por meio da WEB e o processo de persistência, que significa a gravação em meio não volátil dos mesmos, recebidos pelo cliente WEB.

As solicitações de serialização de objetos para a troca de conteúdo entre parceiros de negócio podem se tornar automatizadas pela utilização de um programa de interface pelo Cliente WEB. Esta proposta define este programa como responsável pela automatização dos processos no lado cliente WEB, que são a solicitação, o recebimento, a extração e a persistência local do objeto documento estruturado.

Esta proposta de aplicação compõe uma solução integrada que envolve, de um lado, a responsabilidade pela serialização de objetos utilizando marcações textuais como forma de guardar a estrutura dos objetos, resultando em um documento estruturado, por parte dos servidores WEB, e de outro, a habilidade de tornar disponível uma interface na máquina cliente WEB capaz de automatizar os procedimentos de solicitação de serialização de objetos e de gravação de arquivos contendo os objetos serializados em forma de documentos estruturados.

Esta aplicação integrada viabiliza que outros processos do cliente possam ser realizados sobre o conteúdo coletado e gravado automaticamente, interpretando e extraindo as informações requeridas.

Os processos requeridos para a existência da aplicação WEB integrada são resumidamente enumerados a seguir:

a) **solicitar** a serialização de objetos de forma automática;

b) **serializar** o objeto solicitado;

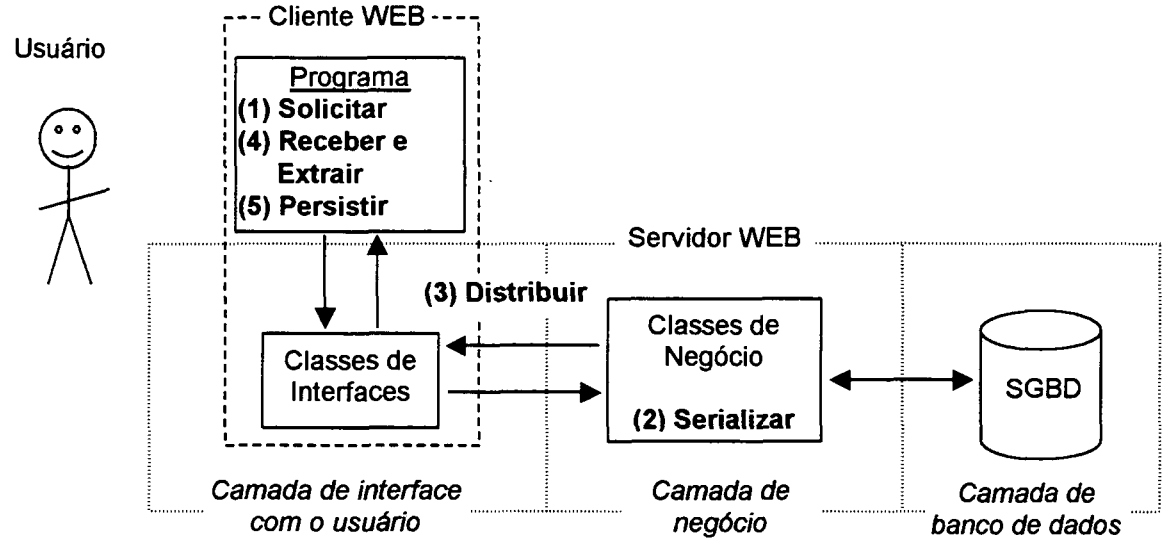
c) **distribuir** o objeto serializado por meio da WEB;

d) **receber** e **extrair** o objeto serializado;

e) **persistir** o objeto serializado.

Estes processos foram contextualizados no ambiente de aplicações WEB, conforme mostrado na figura 8, sendo descritos individualmente na seqüência.

FIGURA 8 - PROCESSOS REQUERIDOS PELA APLICAÇÃO WEB INTEGRADA



3.1.1 Solicitação da Serialização de Objetos de Forma Automática

É o processo realizado por um programa cliente WEB indicado pela proposta de aplicação integrada. É executado a partir da máquina cliente WEB e utiliza os mesmos recursos de rede dos programas navegadores, como o protocolo da camada de aplicação HTTP que usa o conceito de conexão persistente como um padrão em sua última versão e o protocolo de transporte TCP, que é utilizado no ambiente de aplicações WEB por proporcionar a garantia de entrega de dados e a criação de um circuito virtual entre o Cliente WEB e o Servidor WEB.

O programa cliente WEB deve fazer uso de soquetes para realizar a comunicação com o ambiente Internet e informações como o servidor de páginas WEB endereçado, a porta de comunicação TCP/IP utilizada, o nome do processo a ser executado no servidor WEB.

3.1.2 Serialização do Objeto Solicitado

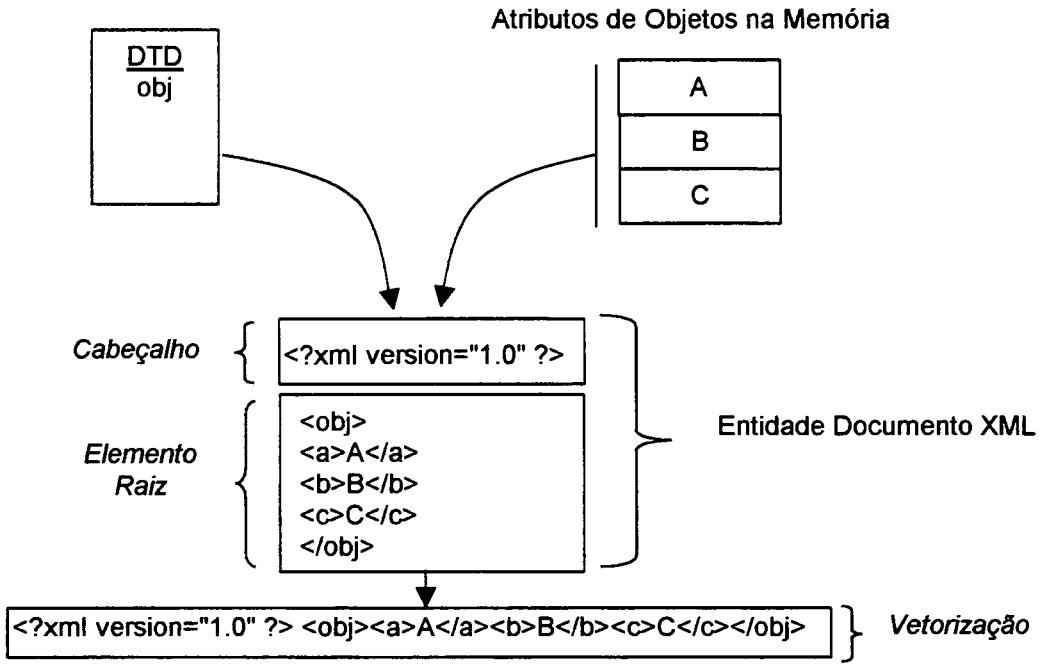
A aplicação WEB integrada utiliza um padrão reconhecido pelo mercado a fim de incluir marcas textuais de estrutura no objeto a ser serializado. Este formato é

proporcionado pelo padrão XML e pela utilização de uma definição DTD associada ao documento XML válido.

A serialização proposta está baseada na conversão completa dos atributos de instâncias de objetos durante seu estado de existência em memória para um vetor de caracteres somados ao formato XML, contendo apenas a estrutura do objeto e seu conteúdo em uma única unidade física, chamada de Entidade Documento. Os atributos de um objeto podem ser mapeados para uma estrutura de árvore, aderindo à hierarquia de dados do formato XML. Os vários tipos de dados devem ser convertidos para o tipo caractere em função da definição de documento XML.

O pré-requisito para o objeto ser serializado no formato de documento válido XML é ter uma definição DTD associada, que possui a definição dos elementos e atributos permitidos para o documento gerado por esta serialização. A figura 9 apresenta a criação do documento XML.

FIGURA 9 - SERIALIZAÇÃO DE OBJETOS UTILIZANDO O FORMATO XML



É necessária a definição do método SerializaXML () para as classes de objetos passíveis de serialização, como pré-requisito para que os mesmos possam ser serializados. Tal método mapeia os atributos de objetos instanciados em

memória para um vetor de caracteres no formato de documento XML e este vetor é deixado disponível para ser enviado para a WEB. A distribuição de operações dos objetos pode ser efetuada apenas quando houver alteração nos procedimentos de operações da classe, ou quando solicitada.

O processo de serialização pode ser executado dinamicamente como resposta de uma requisição.

3.1.3 Distribuição do Objeto Serializado por meio da WEB

É o processo iniciado pelo servidor WEB como resposta a uma requisição HTTP, que envia o objeto documento estruturado para o programa cliente WEB que a requisitou, utilizando o ambiente WEB e Internet.

O processo de distribuição faz uso do protocolo de aplicação HTTP, que conduz a resposta de forma síncrona, sobre o protocolo de transporte TCP, o qual garante a entrega de dados e a conexão persistente.

Um cabeçalho incluído pelo protocolo HTTP é adicionado antes do documento XML. Este cabeçalho contém o estado de retorno desta conexão.

3.1.4 Recebimento e Extração do Objeto Serializado

Este processo é de responsabilidade do programa cliente WEB e faz uso de classes de interface de E/S para receber o objeto documento estruturado.

A resposta gerada pelo servidor WEB possui um cabeçalho requisitado pelo protocolo HTTP que deve ser lido e interpretado por este processo a fim de recuperar o estado de retorno desta resposta. Este estado deve ser enviado para um mecanismo gerenciador de processos. Caso apresente-se com erro, uma mensagem a um gerenciador externo deve ser enviada e o programa deve ser encerrado.

O início do objeto documento estruturado deve ser detectado para sua completa extração, a fim de iniciar o tratamento de persistência descrito a seguir.

3.1.5 Persistência do Objeto Serializado

Este processo é de responsabilidade do programa cliente WEB e realiza a gravação em meio não volátil do objeto serializado recebido do servidor WEB, em um arquivo no formato XML. Para tanto, utiliza-se de um método de gravação de caracteres.

Para a proposta de aplicação integrada, somente documentos estruturados válidos são aceitos, podendo a validação ser comprovada através de *parsers*, como por exemplo, o próprio programa navegador.

3.2 APLICABILIDADE DA PROPOSTA

Os itens anteriores introduziram o conhecimento da arquitetura e recursos que suportam a alternativa proposta. Com o objetivo de observar a aplicabilidade da proposta integrada, um experimento foi realizado com uma pequena aplicação. A descrição do experimento realizado é apresentada no capítulo 4 – Experimento. A seguir são apresentadas algumas considerações sobre esta aplicação.

3.2.1 Considerações sobre a Aplicação

Baseando-se nos processos requeridos pela proposta, nos conceitos apresentados neste capítulo e no anterior, esta aplicação considera:

a) a utilização de um documento estruturado e os conceitos associados de conteúdo e estrutura, sendo que o objeto documento estruturado criado deve ser composto de apenas uma entidade;
b) a criação de uma definição DTD associada ao documento estruturado e referenciada na Declaração do Tipo de Documento;
c) a publicação de um documento estruturado válido, observado com o uso de um *parser* validador;
d) o uso de conexão persistente, de acordo com a última versão do protocolo de aplicação HTTP;
e) a garantia de entrega de dados e da criação de um circuito virtual em função do uso do protocolo de transporte TCP;

f) a identificação das classes de tratamento de E/S orientadas à conexão, garantindo a automatização e a utilização do protocolo de transporte TCP;

g) a utilização da notação *Unified Modeling Language* (UML) como instrumento para modelar a aplicação (conceitos básicos e exemplos de utilização são mostrados no Apêndice 1 e um aprofundamento pode ser obtido em (LARMAN, 1998) (ERIKSSON; PENKER, 1998));

h) a opção pela linguagem de programação Java para sua implementação, em função de sua portabilidade, segurança, suporte a E/S, a caracteres internacionais e o conceito "servlet". Estas características são apresentadas no Apêndice 2;

3.3 CONSIDERAÇÕES FINAIS

Este capítulo identificou os procedimentos necessários para automatizar as atividades de um cliente WEB em busca de conteúdo textual estruturado. Foi ressaltada a importância de marcas textuais no processo de serialização de objetos. Conceitos de serialização, persistência e distribuição de objetos no ambiente de aplicações WEB foram apresentados e os requisitos necessários para a proposta foram definidos e explicados. Para a observação da aplicabilidade da proposta de aplicação integrada, foram definidos a criação e requisitos de um experimento que é apresentado no capítulo 4.

A homogeneidade no tratamento de documentos estruturados transferidos e a independência de plataforma, somados à abrangência mundial da WEB, podem ser características tanto mais desejadas que possam justificar o incremento no tráfego de dados, incremento este causado pela inserção de marcas nos documentos estruturados trafegados em linhas de comunicação.

4 EXPERIMENTO

A validação da proposta integrada de serialização de objetos por meio da WEB utilizando o formato XML foi feita através da elaboração do experimento que consiste da construção e observação de um aplicativo que implementa um subconjunto da proposta definida no capítulo anterior. Este aplicativo foi modelado utilizando a linguagem de modelagem UML. Primeiro, uma seqüência de atividades é enumerada e, para cada uma destas, seu objetivo. Na seqüência, as demais fases de modelagem são citadas juntamente com seus respectivos diagramas. Após, são apresentados os resultados da observação e ao final é feita uma análise apresentada nas considerações finais.

4.1 METODOLOGIA

Inicialmente, as atividades a serem feitas foram enumeradas, sendo elas:

a) escolher um documento estruturado que possua preferencialmente um conjunto de dados diversos e uma disposição hierárquica de estruturas de dados aninhadas, para servir de objeto base durante a modelagem de um sistema;
b) apresentar um caso de uso essencial a fim de descrever as funcionalidades do sistema;
c) elaborar as demais fases de modelagem com seus respectivos diagramas e apresenta-los;
d) criar uma definição DTD para o objeto documento estruturado, após a modelagem de sua classe na fase de projeto, pois esta definição será usada durante a criação e a validação do documento XML;
e) implementar a aplicação;
f) observar os resultados esperados.

O primeiro passo necessário foi a escolha de um documento estruturado que possuísse interações e uma variedade de tipos de dado, e o mesmo sendo abstraído como objeto. Poderia ser qualquer documento, sendo que se optou por

um extrato bancário de conta corrente, pois o mesmo possui tipos de dados como cadeias de caracteres, datas, números inteiros e decimais. É interessante observar que os lançamentos de movimentação de conta corrente são interações que necessitam de preservação da disposição hierárquica das estruturas de dados aninhadas. O exemplo de documento estruturado escolhido para servir de base durante a modelagem é mostrado na figura 10.

FIGURA 10 - DOCUMENTO ESTRUTURADO EXEMPLO ESCOLHIDO PARA A MODELAGEM

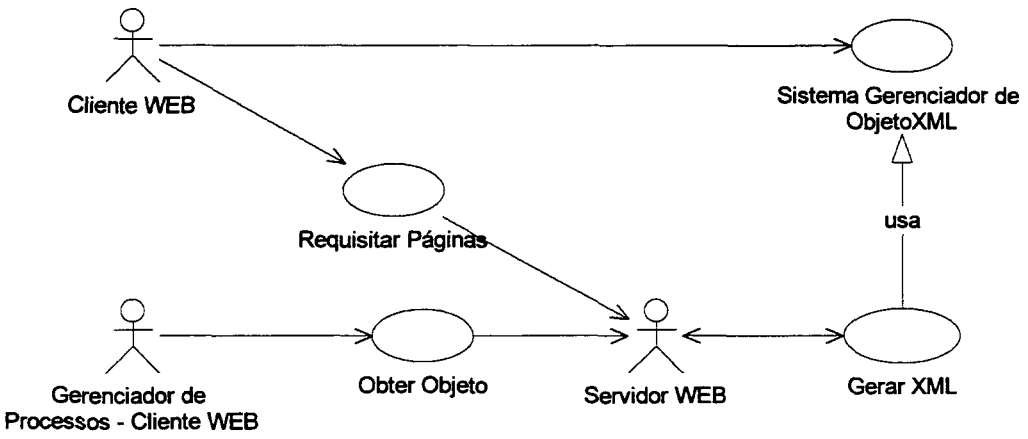
```
                          BANCO BRASILEIRINHO S/A

       Extrato de Conta Corrente
 Banco: 85    Ag: 002    Conta: 01234-5     Cliente: Asdrubal Oliveto Neto
        Lancamentos do periodo de 11/10/2000 A 19/10/2000
          Saldo Anterior.......              2.470,48 C
       Data  Documento Descricao         Valor(D/C)            Saldo(D/C)
-------------------------------------------------------------------------

  11/10 540649  Cheque Compensado            59,00 D
        700151  Rend Poup Integrad           0,17 C            2.411,65 C
  16/10 700151  Rend Poup Integrad           0,78 C
        700704  Debito Iptu                 48,80 D            2.363,63 C
  18/10 700151  Deposito                    18,37 C            2.382,00 C
  19/10 700151  Deposito                    14,23 C
        700701  Fatura Telepar              59,12 D
        701722  Telepar Celular             33,90 D
        540650  Cheque Compensado           48,00 D
        700151  Rend Poup Integrad           0,42 C            2.255,63 C

 Sujeito A Alteracoes Por Valores Em Transito
```

4.2 MODELAGEM UML

Os modelos auxiliam a visualização e permitem a especificação da estrutura ou comportamento de um sistema, fornecendo também linhas mestras que guiam a construção de um sistema e documentam as decisões tomadas (BOOCH; RUMBAUGH; JACOBSON, 1997, p. 4-9). Este experimento será apresentado na seqüência através da linguagem de modelagem UML.

O diagrama de caso de uso essencial, mostrado na figura 11, apresenta dois atores principais, **Cliente WEB** e **Gerenciador de Processos - Cliente WEB**, chamados de ativos pois iniciam casos de uso do sistema, e o ator **Servidor WEB** que representa o Sistema Servidor WEB.

FIGURA 11 - DIAGRAMA DOS CASOS DE USO



São apresentados os casos de uso **Requisitar Páginas**, **Obter Objeto, Gerar XML** e **Sistema Gerenciador de ObjetoXML**. Este último representa genericamente o processo de gerenciamento de objetos passíveis de serialização, que neste experimento é o Sistema de Gerenciamento de Operações Bancárias. O **Cliente WEB** inicia o caso de uso **Sistema Gerenciador de ObjetoXML** efetuando transações bancárias em suas operações. A operação representa um contrato com a instituição bancária "Brasileirinho S/A", por exemplo, através de uma conta corrente. Um objeto serializado em forma de documento estruturado XML pode ser solicitado pelo ator **Gerenciador de Processos - Cliente WEB**, o qual inicia o caso de uso **Obter Objeto**.

Os casos de uso irão interagir com o Sistema Servidor WEB, que está presente através do estereótipo de ator **Servidor WEB** e que inicia o caso de uso **Gerar XML** a partir de uma mensagem recebida de **Obter Objeto**.

Este experimento tem como objetivo observar a aplicabilidade da proposta deste trabalho através da implementação dos casos de uso **Obter Objeto** e **Gerar**

XML, e da implementação parcial das funcionalidades do objeto exemplo "Operação Bancária", existente no caso de uso **Sistema Gerenciador de ObjetosXML.**

4.2.1 Diagramas da Fase de Análise

Para este modelo, utilizaram-se sete classes, sendo elas: **Cliente WEB**, **Operação**, **Lançamentos**, **Tipo de Lançamento**, **Natureza do Lançamento**, **HttpServlet** e **Soquete**; e quatro generalizações: **Conta**, **Conta Corrente**, **Conta Poupança**, **ServletXML**, representadas no Diagrama de Classes do modelo conceitual mostrado pela figura 12.

O diagrama pode ser descrito da seguinte maneira: um **Cliente** pode possuir uma ou mais **Operações** (1..*), que representam contratos existentes entre o cliente e a instituição bancária. Estas **Operações** podem conter zero ou mais (0..*) **Lançamentos**. Neste diagrama é mostrado que a classe **Operação** possui a generalização **Conta**, que por sua vez, possui duas outras generalizações, **Corrente** e **Poupança**. Os **Lançamentos** possuem um **Tipo de Lançamento**, e um **Tipo de Lançamento** possui uma **Natureza do Tipo de Lançamento**.

Também é mostrado que a classe **HttpServlet** possui a generalização **ServletXML** e é apresentada uma classe **Soquete.**

A figura 13 apresenta o Diagrama de Seqüência modelado durante a fase de análise. O sistema é tratado como um único bloco, sendo que o ator **Gerenciador de Processos – Cliente WEB** envia a mensagem **Solicita_Objeto ()**, a qual inicia o sistema, especificamente o caso de uso **Obter Objeto** na camada cliente, e recebe ao final da execução uma mensagem, **Envia_Estado ()**, a qual informa a condição de término do Sistema.

FIGURA 12 – DIAGRAMA DE CLASSES

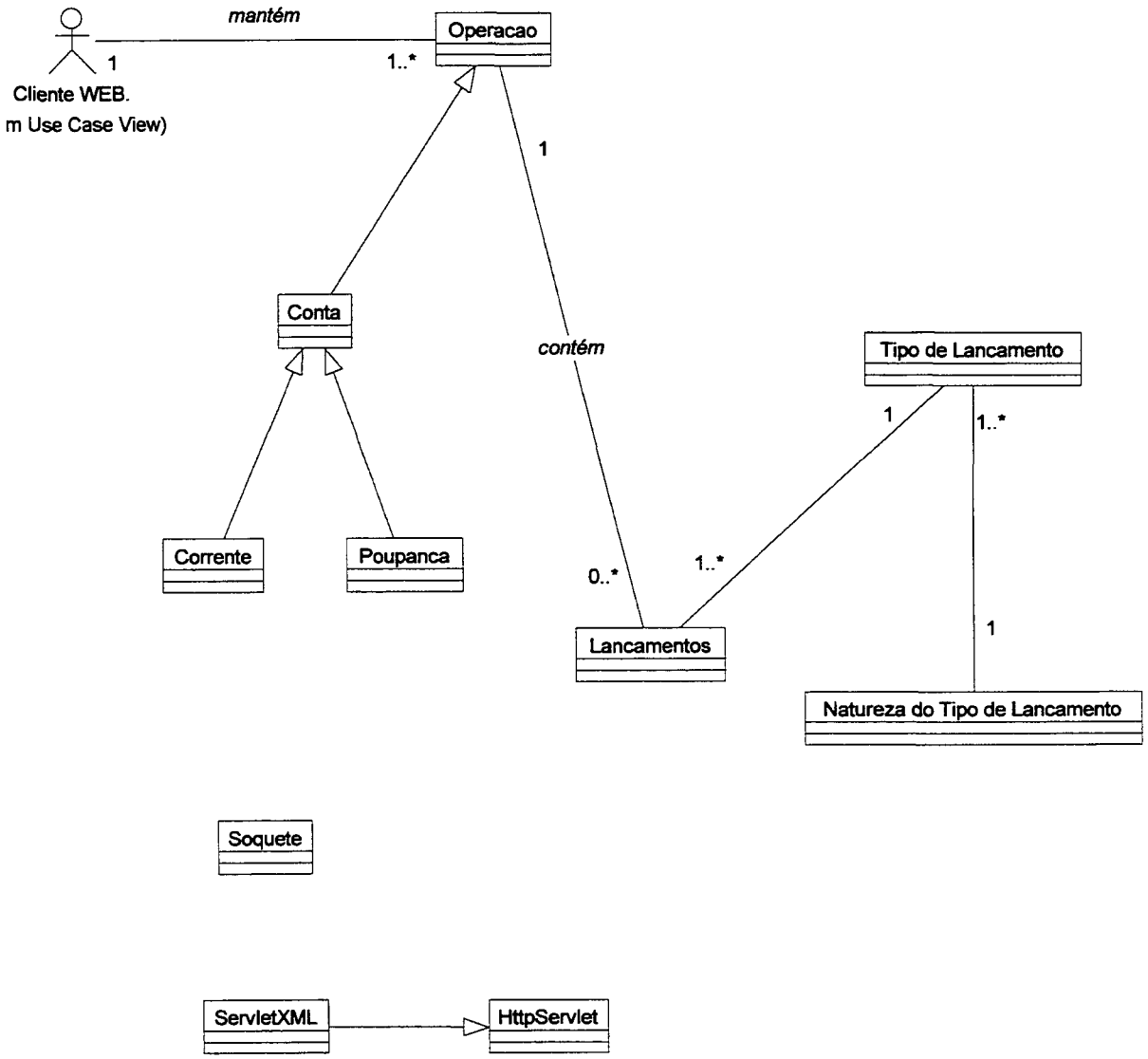
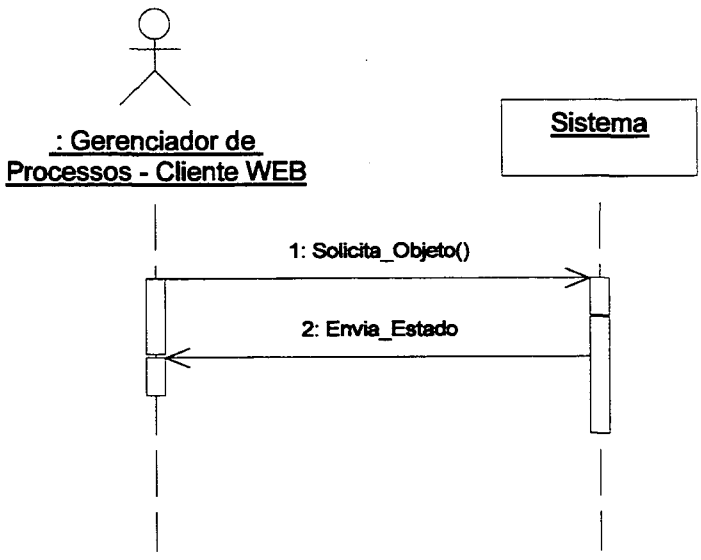


FIGURA 13 – DIAGRAMA DE SEQUÊNCIA DA FASE DE ANÁLISE



4.2.2 Diagramas da Fase de Projeto

A modelagem do Diagrama de Caso de Uso Real é a primeira atividade da fase de projeto. Este diagrama fornece uma visão concreta de como o caso de uso será realizado. A seguir são apresentados os casos de uso real ***Obter Objeto*** (figura 14) e ***Gerar XML*** (figura 16) e, associado a cada um destes, o Diagrama do Curso Típico dos Eventos (figuras 15 e 17).

FIGURA 14 - CASO DE USO REAL OBTER OBJETO

Caso de uso:	**Obter Objeto**
Ator:	Gerenciador de Processos – Cliente WEB (inicia)
Propósito:	Obter o objeto do documento estruturado no formato XML
Descrição:	O ator Gerenciador de Processos – Cliente WEB inicia este caso de uso, ativando a classe Soquete, que, após seu processamento, retorna uma mensagem informando sobre a condição seu término.

FIGURA 15 - CURSO TÍPICO DOS EVENTOS DO CASO DE USO OBTER OBJETO

Ações de Atores	**Respostas do Sistema**
1) Início - O ator Gerenciador de Processos – Cliente WEB ativa a classe Soquete	
	2) A Classe Soquete solicita conexão ao Servidor WEB.
3) O ator Servidor WEB aceita a conexão, devolve a condição de aceite e o curso segue para o passo 4; ou o Servidor WEB recusa a conexão, devolve a condição de recusa e o curso segue para o passo 7.	
	4) A classe Soquete envia uma requisição HTTP ao Servidor WEB solicitando a execução da classe ServletXML, passando os parâmetros de identificação do objeto requisitado.
5) O Servidor WEB ativa a classe ServletXML e encaminha a resposta recebida através da conexão HTTP (**distribuição**).	
	6) A classe Soquete recebe a resposta de sua

	requisição e, caso tenha recebido um código de erro, o repassa para o Gerenciador de Processos Processos – Cliente WEB e o curso segue para 7; caso contrário, trata a mensagem recebida, *armazena o objeto* (**persistência**) e repassa a condição de término para o Gerenciador de Processos – Cliente WEB e o curso segue para 7.
7) O ator Gerenciador de Processos – Cliente WEB recebe a condição de término de execução da classe Soquete.	

FIGURA 16 - CASO DE USO REAL GERAR XML

Casos de uso:	**Gerar XML**
Ator:	Servidor WEB (inicia)
Propósito:	Obter o objeto do documento estruturado no formato XML
Descrição:	O ator Servidor WEB inicia este caso de uso, ativando a classe ServletXML que processa a requisição HTTP e devolve como resposta o objeto serializado no formato XML após o cabeçalho enviado pelo protocolo HTTP.

FIGURA 17 - CURSO TÍPICO DOS EVENTOS DO CASO DE USO GERAR XML

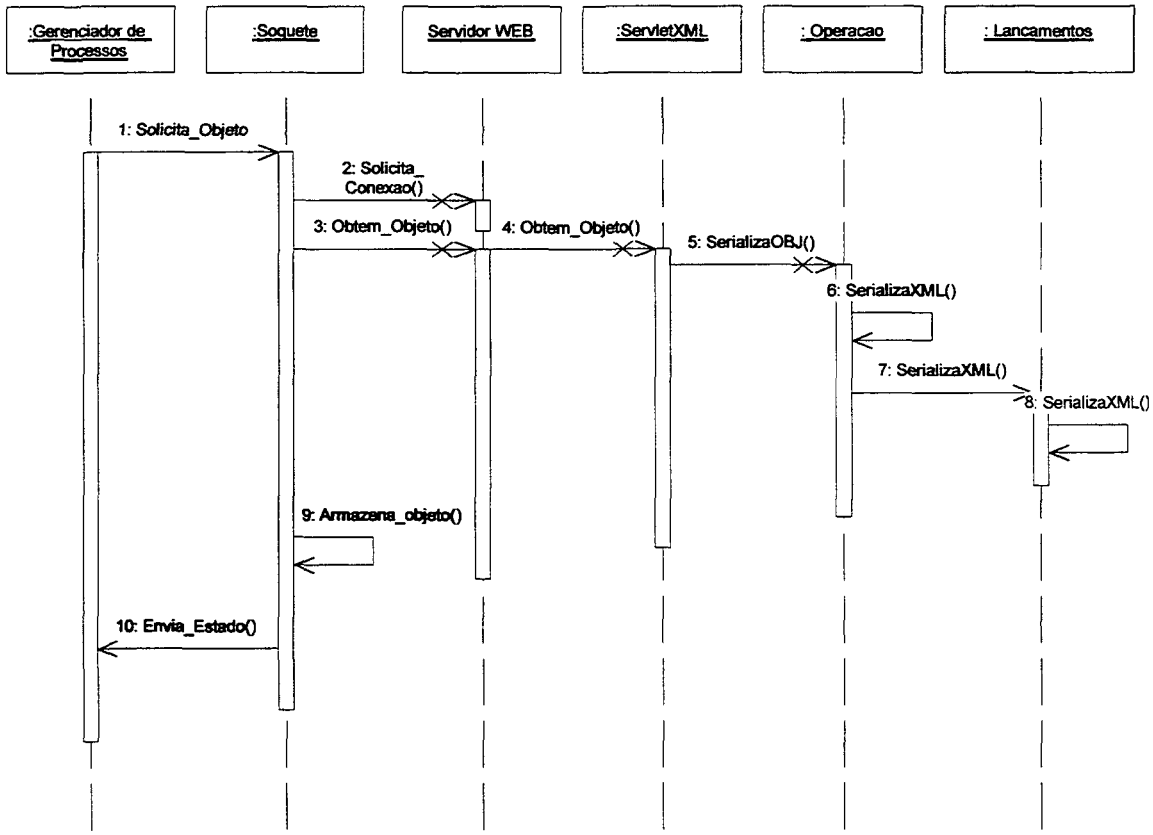
Ações de Atores	**Respostas do Sistema**
1) O ator Servidor WEB ativa a classe ServletXML.	
	2) A classe ServletXML obtém e valida os parâmetros recebidos, e, caso os mesmos não sejam válidos, ou caso não encontre o objeto requisitado, devolve como resposta o erro e o curso segue para 3; caso os parâmetros sejam válidos e o objeto exista, efetua a **serialização** do objeto requisitado e devolve como resposta o vetor de caracteres gerado.
3) O Servidor WEB encaminha a resposta através da conexão HTTP (**distribuição**).	

Os diagramas de Curso Típico dos Eventos mostram que o processo de serialização é uma resposta dada pelo sistema através da classe *ServletXML*, o

processo de distribuição é feito pelo *Servidor WEB* e o processo de persistência é uma resposta dada pelo sistema através da classe *Soquete.*

Na figura 18 é apresentado o Diagrama de Seqüência de Eventos, o qual ilustra a interação dos objetos com base na seqüência de mensagens trocadas entre eles, dentro do tempo.

FIGURA 18 - DIAGRAMA DE SEQÜÊNCIA DE EVENTOS

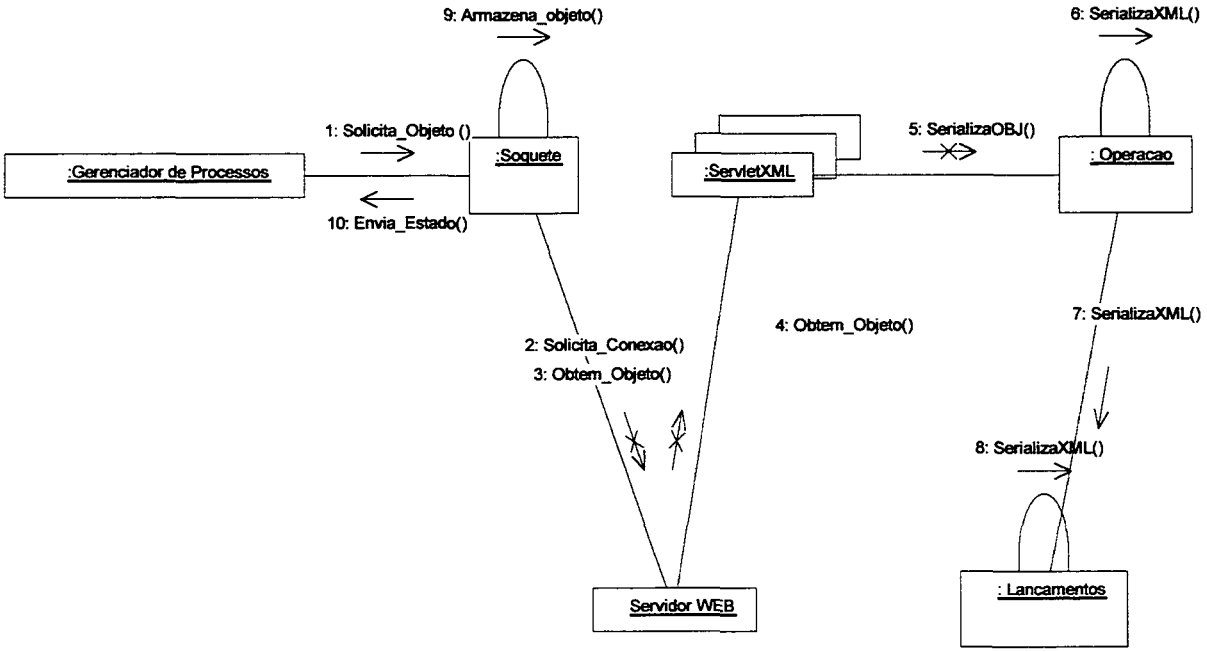


Pode-se observar no diagrama da figura 18 que a mensagem *1:Solicita_Objeto()* enviada pelo gerenciador de processo é assíncrona, inicia a classe *Soquete* e libera o gerenciador, que receberá uma mensagem de término *10: Envia_Estado()*, contendo o estado final da execução do processo enviada pela classe *Soquete.*

A classe Soquete que implementa o modelo requisição-resposta HTTP envia mensagens síncronas para o servidor WEB. A primeira, solicitando conexão, e a segunda, solicitando o objeto a ser serializado.

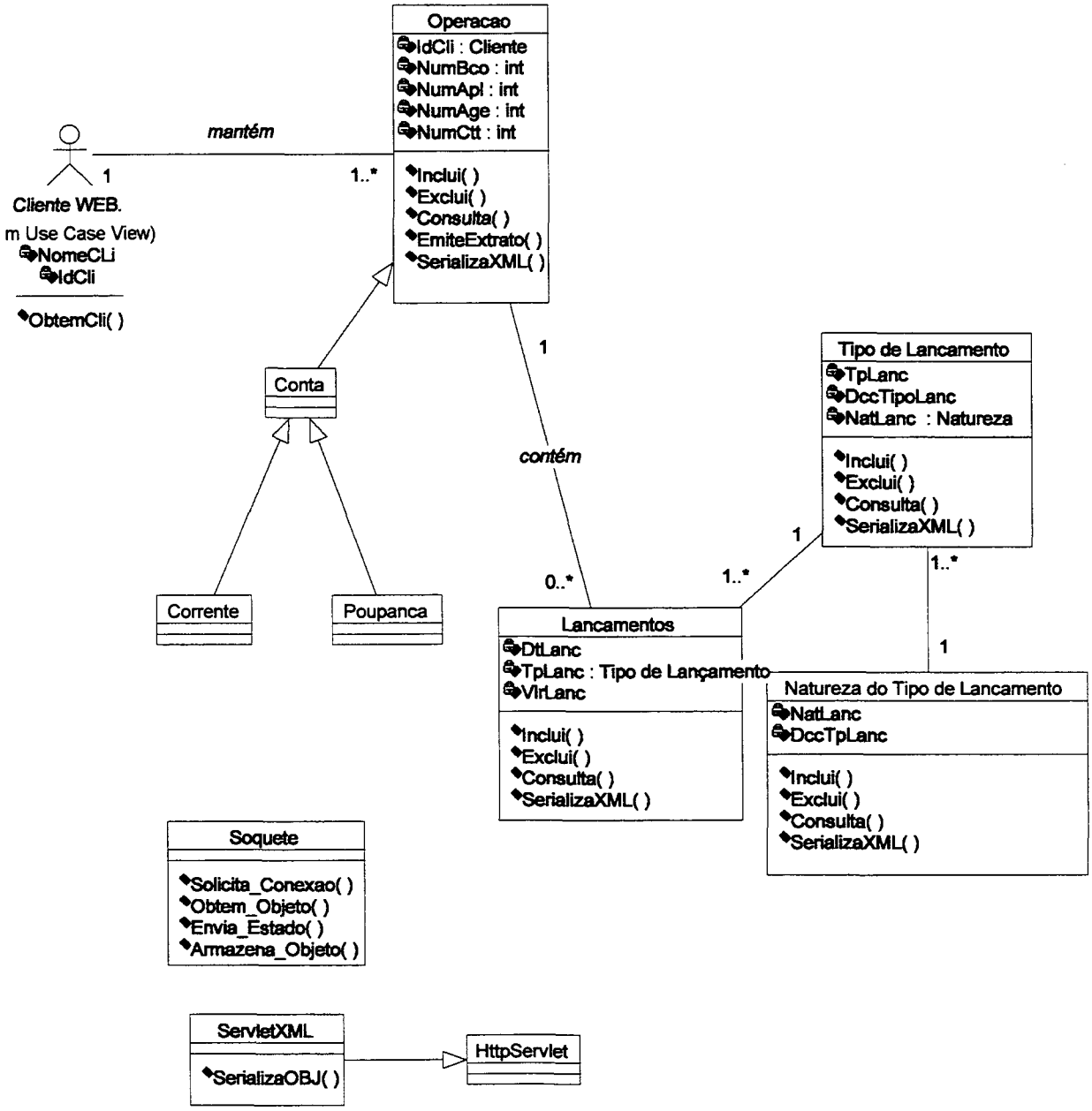
As interações no espaço dos objetos atuais e suas ligações de colaboração são modeladas a partir do Diagrama de Colaboração, que é mostrado na figura 19.

FIGURA 19 - DIAGRAMA DE COLABORAÇÃO



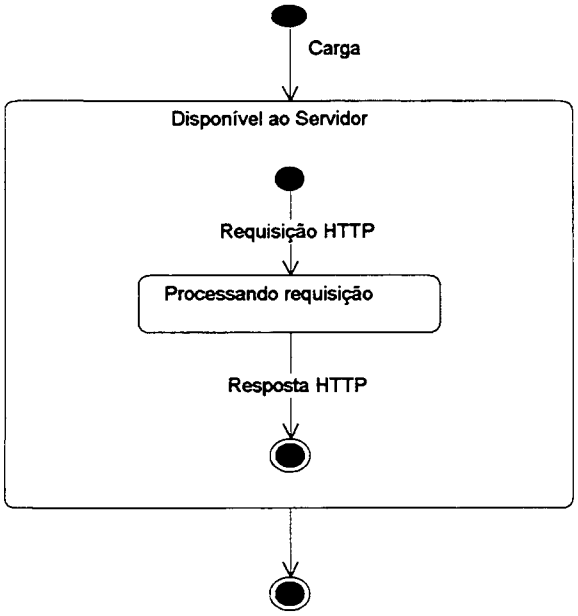
Na fase de projeto ocorre o refinamento da modelagem de classes da fase de análise, e deve-se incluir os atributos e operações necessárias às classes, como é mostrado na figura 20.

FIGURA 20 – DIAGRAMA DE CLASSES



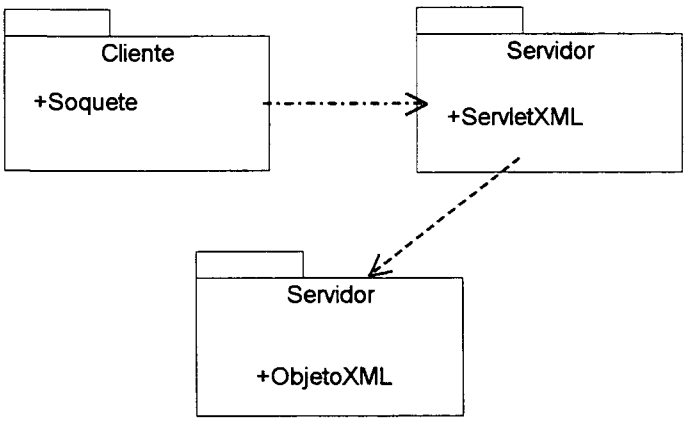
A classe ***ServletXML*** é um processo suportado pelo servidor WEB que, após sua carga, pode responder a várias requisições HTTP, e o diagrama de estado mostrado pela figura 21 apresenta o comportamento de um objeto ativo desta classe.

FIGURA 21 – DIAGRAMA DE ESTADO DO SERVLETXML



A fim de implementar o empacotamento deste experimento, utilizou-se a estrutura mostrada pela figura 22.

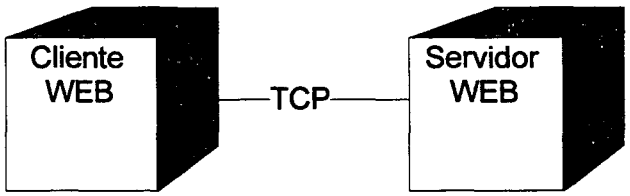
FIGURA 22 – DIAGRAMA DE PACOTES



O componente *ObjetoXML* do diagrama de pacotes da figura 22 representa, de forma genérica, as classes a serem tratadas pelo processo de serialização. Este processo requer a existência do método **SerializaXML()** associado às classes passíveis de serialização.

O Diagrama de Implantação (figura 23) mostra a distribuição de processos e componentes em nós de processamento. O pacote Soquete residirá no Cliente WEB, e os pacotes ObjetoXML e ServletXML residirão no Servidor WEB.

FIGURA 23 - DIAGRAMA DE IMPLANTAÇÃO



4.3 CRIAÇÃO DA DEFINIÇÃO DTD

Associadas à criação de DTD's existem muitas discussões dentro da comunidade XML. MARCHAL cita em (2000, p. 111, 383-384) que a escolha de atributos ou elementos é um debate, pois ambas as técnicas possuem argumentos contra e a favor. RAMALHO sugere em (2000) que esta discussão seja resolvida caso a caso, e apresenta alguns princípios que podem auxiliar a decisão, como, por exemplo, definir sempre como "elemento" a informação considerada estrutural, relacionada a aspectos visuais, ou que possua processamento especial, e definir sempre como "atributo", a informação que qualifique o conteúdo ou faça parte de um padrão repetitivo (como uma lista enumerada, por exemplo).

Para a elaboração deste experimento, todas as informações foram consideradas estruturais, sujeitas a processamentos especiais e todos os dados do documento foram tratados como "elementos".

A definição DTD resultante desta análise é mostrada na figura 24.

FIGURA 24 – DEFINIÇÃO DTD PARA UM EXTRATO BANCÁRIO

```
<!–XML DTD–>
<!– DTD para um Extrato Bancario - por Edilza Romanichen - 2001.01.19 –>
<!– –>
<!ELEMENT Extrato (Cliente+,Operacao)>
<!– –>
<!ELEMENT Cliente (#PCDATA)>
<!– –>
<!ELEMENT Operacao (NumBanco, NumAplicacao, NumAgencia, NumContrato, PeriodoIni, PeriodoFin, SaldoAnt, Lancamentos+,SaldoFin)>
<!– –>
<!ELEMENT NumBanco (#PCDATA)>
<!ELEMENT NumAplicacao (#PCDATA)>
<!ELEMENT NumAgencia (#PCDATA)>
<!ELEMENT NumContrato (#PCDATA)>
<!ELEMENT PeriodoIni (#PCDATA)>
<!ELEMENT PeriodoFin (#PCDATA)>
<!– –>
<!ELEMENT SaldoAnt (VlrSaldoAnt,NatSaldoAnt)>
<!– –>
<!ELEMENT VlrSaldoAnt (#PCDATA)>
<!ELEMENT NatSaldoAnt (#PCDATA)>
<!– –>
<!ELEMENT Lancamentos (DtLanc,TpLanc,DccTpLanc,VlrLanc,NatLanc)+>
<!– –>
<!ELEMENT DtLanc (#PCDATA)>
<!ELEMENT TpLanc (#PCDATA)>
<!ELEMENT DccTpLanc (#PCDATA)>
<!ELEMENT VlrLanc (#PCDATA)>
<!ELEMENT NatLanc (#PCDATA)>
<!– –>
<!ELEMENT SaldoFin (VlrSaldoFin,NatSaldoFin)>
<!– –>
<!ELEMENT VlrSaldoFin (#PCDATA)>
<!ELEMENT NatSaldoFin (#PCDATA)>
```

4.4 RECURSOS UTILIZADOS NA IMPLEMENTAÇÃO DA APLICAÇÃO

O sistema operacional utilizado foi o *Windows 2000 Server*, da empresa *Microsoft*. Para tornar disponível a utilização do protocolo HTTP e *servlet* no experimento, foi utilizado o servidor WEB *JavaWebServer 2.0*, da empresa *Sun Microsystem*, com suporte a *servlets*.

Os programas empregaram a linguagem Java e importaram classes da biblioteca de classes *Java Development Kit* (JDK) 1.1.8.

A modelagem deste experimento foi feita em UML com o produto *Rational Rose/C++* Demo, versão 4.0.3, da empresa *Rational Software Corp.*. Esta versão de demonstração é limitada a trinta classes e atores e a 10 estados, apresentando-se suficiente para a modelagem deste experimento.

4.5 RESULTADOS OBTIDOS

Para a verificação do experimento, o ator *Gerenciador de Processos* foi considerado como uma chamada de execução da classe Java em uma linha de comando do *prompt do DOS*.

A classe *Soquete* foi utilizada a partir de uma máquina cliente. Inicialmente, foi requisitada uma página estática para verificar as funcionalidades da classe. Sua execução foi realizada através de chamada pelo aplicativo Java e os resultados de gravação de arquivo e devolução de estado foram verificados, obtendo como resultado o documento estruturado XML. Posteriormente, os mesmos resultados foram verificados através da execução da requisição da classe *ServletXML*.

Foi necessário carregar a classe *ServletXML* no Servidor WEB com suporte a *Servlets* a fim de torná-la disponível para a execução. Este procedimento é necessário para efetivar as manutenções/alterações desta classe, deixando-a atualizada no servidor WEB. O servidor permite que o *servlet* seja carregado manualmente sob demanda ou durante a inicialização do servidor. A classe *ServletXML* foi utilizada através de uma solicitação via programa navegador, onde as funcionalidades de geração de página dinâmica XML puderam ser observadas.

O método *SerializaXML* da classe *Operação* foi observado inicialmente gerando um arquivo local ao servidor WEB e posteriormente recebeu a referência da classe *PrintWriter* (passada pela classe *ServletXML*) a qual representa a conexão HTTP. Ambas as utilizações geraram a cadeia de caracteres esperada, criando um documento estruturado "Extrato Bancário", sendo assim observadas as funcionalidades do método *SerializaXML*.

Todas as observações feitas nas classes modeladas de forma independente e integrada obtiveram os resultados esperados.

Na figura 25 é mostrada a visualização do documento "Extrato.xml" a partir de um programa navegador.

FIGURA 25 - "EXTRATO.XML" VISUALIZADO A PARTIR DE UM PROGRAMA NAVEGADOR

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Extrato (View Source for full doctype...)>
<!-- Nome do Arquivo: Extrato.xml  -->
- <Extrato>
    <Cliente>Asdrubal Oliveto</Cliente>
  - <Operacao>
      <NumBanco>85</NumBanco>
      <NumAplicacao>12</NumAplicacao>
      <NumAgencia>002</NumAgencia>
      <NumConta>01234-5</NumConta>
      <PeriodoIni>2000.10.11</PeriodoIni>
      <PeriodoFin>2000.10.19</PeriodoFin>
    + <SaldoIni>
    + <Lancamentos>
    + <Lancamentos>
    + <Lancamentos>
    + <Lancamentos>
    - <SaldoFin>
        <VlrSaldoFin>2255,63</VlrSaldoFin>
        <NatSaldoFin>C</NatSaldoFin>
      </SaldoFin>
    </Operacao>
  </Extrato>
```

4.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou um experimento que foi realizado com o objetivo de observar a adaptabilidade da proposta de aplicação WEB. Foi utilizado um documento estruturado como exemplo. Houve a definição e implementação de classes que permitiram a observação da aplicação proposta sugerida neste texto. A implementação realizada tomou como base um documento exemplo e foi modelada em UML. Uma definição DTD foi construída a partir dos atributos do objeto, com a finalidade de estabelecer o tipo deste documento, para que o mesmo possa ser verificado por *parsers* validadores da meta-linguagem XML.

Observou-se que a construção de uma definição DTD a partir de um objeto modelado pode ser considerada trivial, pois ambos podem ser representados com uma estrutura hierárquica. Durante a elaboração da definição DTD do experimento foi observado que outras considerações sobre esta construção devem ser objeto de estudo, como por exemplo, a decisão sobre a definição de elementos ou atributos,e a preocupação com a flexibilidade da definição DTD para acomodar adaptações futuras, em caso de evoluções necessárias.

É possível, implementando o método Desserializa(), transformar o documento XML recebido em atributos de objetos guardados em um banco de dados, caso desejado.

Este experimento demonstrou a aplicabilidade da alternativa da solução proposta nesta dissertação.

O Apêndice 3 mostra uma seqüência de passos para que a reprodução deste experimento possa ser obtida.

5 CONCLUSÃO

Este trabalho propôs uma aplicação WEB integrada como alternativa para solucionar o problema de coleta de conteúdo textual publicado na WEB, utilizando como recursos a serialização de objetos. Esta aplicação WEB integrada foi definida com base na execução de processos necessários na máquina cliente e no servidor WEB.

Foram citados os trabalhos relacionados a documentos estruturados e fornecida uma visão sucinta de definições e recursos associados ao padrão XML. O padrão XML foi escolhido para resolver o problema de incompatibilidade entre os diversos padrões de estruturação de informações na WEB.

Os procedimentos necessários para automatizar as atividades de um cliente WEB em busca de conteúdo textual estruturado foram identificados. Foi ressaltada a importância de marcas textuais no processo de serialização de objetos. Conceitos de serialização, persistência e distribuição de objetos no ambiente de aplicações WEB foram apresentados e os requisitos necessários para a proposta foram definidos e explicados. Para a observação da aplicabilidade da proposta de aplicação integrada, foram definidos a criação e requisitos de um experimento que é apresentado no capítulo 4.

Um experimento, modelado em UML, foi conduzido com a finalidade de observar a adaptabilidade da aplicação proposta. Foi utilizado um documento estruturado como exemplo. Houve a definição e implementação de classes que permitiram a observação da aplicação proposta sugerida neste texto. Uma definição DTD foi construída a partir dos atributos do objeto, com a finalidade de estabelecer o tipo deste documento, para que o mesmo possa ser verificado por *parsers* validadores da meta-linguagem XML.

A introdução do conhecimento da arquitetura e recursos que suportam a alternativa proposta gerou subsídios para o refinamento das alternativas tecnológicas apresentadas e permitiu a escolha do protocolo de transporte TCP em função da garantia de entrega de dados e da criação de um circuito virtual; a opção pela conexão persistente do protocolo HTTP recomendada em sua versão mais

recente; a identificação das classes de tratamento de entradas e saídas orientadas à conexão, o que capacitou a automatização da solicitação feita pela máquina cliente, produzida pela criação de um programa que utiliza rotinas genéricas chamadas "soquetes".

A experimentação comprovou o funcionamento das partes modeladas no ambiente cliente e no ambiente servidor, permitindo a comunicabilidade integral entre as partes. Este experimento demonstrou a aplicabilidade da alternativa da solução proposta nesta dissertação e mostrou que esta solução é uma alternativa na transferência eletrônica de objetos serializados e distribuídos por meio da WEB.

Esta proposta de aplicação integra diferentes tecnologias, como WEB, Internet, serialização de objetos, XML, entre outros, sendo independente de fabricantes e linguagens. Portanto, todos os objetivos deste trabalho foram atingidos.

A utilização desta proposta pode facilitar a interoperabilidade entre domínios específicos de negócios, possibilitando a troca automática e compatível de conteúdos entre pessoas e / ou organizações. O documento estruturado resultante da serialização de objetos facilita o resgate, a reutilização, a interpretação, a localização e o compartilhamento de dados.

Processos computacionais poderiam ser executados após a coleta de conteúdos, e se beneficiarem desta estrutura, interpretando e extraindo as informações requeridas de forma automática.

A informação recebida pelo cliente pode ainda ser validada por *parsers* disponíveis no mercado, sendo independente do processo que a originou.

5.1 PERSPECTIVAS

Aspectos de segurança, que são compostos por mecanismos que suportam o tratamento de informações confidenciais e também a integridade, autenticação e assinatura digital, podem ser incorporados como funcionalidades a esta proposta.

A proposta deste trabalho abrange diversos tipos de máquinas clientes, que utilizando os protocolos HTTP e TCP/IP ou IPng, podem ser adaptados ao ambiente WEB, como por exemplo, assistentes pessoais, celulares, entre outros. Um trabalho

de construção do programa de interface pode permitir esta adaptação sem muito esforço.

REFERÊNCIAS BIBLIOGRÁFICAS

APPARAO, V.; BYRNE, S.; CHAMPION, M.; ISAACS, I.; JACOBS, I.; LE HORS, A.; NICOL, G.; ROBIE, J.; SUTOR, R.; WILSON, C.; WOOD, L.. **Document Object Model (DOM) Level 1 Specification**. World Wide Web Consortiun (W3C), Out. 1998. Disponível em: <http://www.w3c.org/TR/1998/REC-DOM-Level-1-19981001> Acesso em: 24 nov. 2000.

BANKSTON; SEIFERT. **Java 1.1 Unleashed**. Capítulo 31 - Persistence and Java Serialization. 1997. Disponível em: <http://kitap.selcuk.edu.tr/java/Java_1.1_Unleashed/htm/toc.htm> Acesso em: 15 jan. 2001.

BEN-NATAN, R.. **Objects on the Web – Design Building, and Deploying Object-Oriented Applications for the Web**. New York : McGraw-Hill, 1997.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.. **The Unified Modeling Language User Guide.**. Massachusetts : Addison-Wesley, p: 6- , 1999. ISBN 0-201-57168-4.

BRAY, T.; PAOLI, J.; SPERBERG-McQUEEN, C.M. **Extensible Markup Language (XML) 1.0**. World Wide Web Consortiun (W3C), Fev. 1998. Disponível em: <http://www.w3c.org/TR/1998/REC-xml-19980210> Acesso em: 24 nov. 2000.

BRAY, T.; PAOLI, J.; SPERBERG-McQUEEN, C.M.; MALER, E. **Extensible Markup Language (XML) 1.0 (Second Edition)**. World Wide Web Consortiun (W3C), Out. 2000. Disponível em: <http://www.w3c.org/TR/2000/REC-xml-20001006> Acesso em: 24 nov. 2000.

W3C. **Consórcio W3C** . Disponível em: <http://www.w3c.org> Acesso em: 24 nov. 2000.

ERIKSSON, H.; PENKER, M.. **UML Toolkit.** New York : John Wilwy & Sons, INC, p: 01-43, 119-196, 1998.

FORJAN, M. **IPng: Internet Protocol Next Generation**. Brigham Young University – Hawaii Campus, 199?. Disponível em: <http://www.ee.siue.edu/~mforjam/projects/ee580.html> Acesso em: 15 jan. 2001.

GLUSHKO, R. J.; TENEMBAUN, J. M.; MELTZER, B.. An XML Framework for Agent-based E-commerce. **Revista Communications of the ACM.** New York : ACM Press, p: 106-114, mar. 1999.

IBM Corp. **TCP/IP Tutorial and Technical Overview**. Disponível em: <http://www.redbooks.ibm.com> Acesso em: 11 dez. 2000.

KHARE, R.; RIFKIN, A.. XML: A Door to Automated Web Applications. **Revista IEEE Internet Computing.** New York : Computer Society, p: 78-87, ago. 1997a.

_____. Capturing the State of Distributed Systems with XML. **World Wide Ewb Journal Special Issue on XML**, Volume 2, Number 4, p:207-218, dez. 1997b. <http://www.cs.caltech.edu/~adam/papers/xml/xml-for-archiving.html> Acesso em 23 jan. 2001.

LARMAN, C.. **Applying UML and Patterns.** New Jersey : Prentice Hall PTR. 1998.

LESK, M.. **Pratical Digital Libraries.** San Francisco : Morgan Kaufmann. 1997.

MARCHAL, B.. **XML Conceitos e Aplicações.** Tradução Daniel Vieira. São Paulo : Berkeley Brasil, p: 77-121, 347-387, 2000.ISBN 85-7251-564-X.

McGRATH, S.. **XML by Example – Building E-Commerce Applications.** New Jersey : Prentice-Hall. 1998.

MARUYAMA, H.; TAMURA, K.; URAMOTO, N.. **XML and JAVA : Developing WEB Applications**. Massachusetts : Addison-Wesley. 1999.

MSDN Library. **Three-Tier Application Development.** Microsoft Developer Network Library, jul. 1999a. 1 CDROM.

_____. **Serialization.** Microsoft Developer Network Library, jul. 1999b. CDROM.

PIANESSO, A. C. F.; CASTANHO, C. L. O.. **Persistência em Linguagens Orientadas a Objetos**. Universidade Federal de Santa Catarina, 2000. Disponível em: <http://www.ee.siue.edu/~mforjam/projects/ee580.html> Acesso em: 15 jan. 2001.

RAMALHO, J. C. L.. **Anotação Estrutural de Documentos e sua Semântica**. Departamento de Informática, Escola de Engenharia, Universidade do Minho, Tese de Doutorado, 2000. Disponível em: <http://orunner.di.uminho.pt/~jcr/DOCS/phd/src/x3658.htm> Acesso em: 23 jan. 2001.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W.. **Modelagem e Projetos Baseados em Objetos**. 1ª Edição. Rio de Janeiro : Campus, p: 64, 1994. ISBN 85-7001-841-X.

ROBERTS, S.; HELLER, P.; ERNEST, M.. **Complete Java 2 Certification – Study Guide.**. 2ª Edição. San Francisco : Sybex, p: 437-473, 535-574, 2000. ISBN 0-7821-2825-4.

SOARES, L. F. G.; LEMOS, G.; CPLCHER, S.. **Redes de Computadores – Das LANs MANs e WANs às Redes ATM.** Rio de Janeiro : Campus, p: 311-395, 1997.

SUTTON, M. J. D.. **Document Management for the Enterprise: Principles, Techniques, and Applications.** New York : Wiley Computer Publishing. 1996.

USDIN, T.; GRAHAM, T.. XML: Not a Silver Bullet, But a Great Pipe Wrench. **StandardView** Vol.6, N. 3, September 1998, p. 125-133. ACM.

WONG, C.. **HTTP Pocket Reference** . Sebastopol : O`Reilly & Associates. 2000. ISBN 1-56592-862-8.

DOCUMENTOS CONSULTADOS

GROSS, N.; MOELLER, M.. A 'Rosetta Stone' for the WEB? **Revista Business Week.** New York : McGraw-Hill, p: 122-123, jun. 1999.

GRASSMANN, W. K.; TREMBLAY, J.. **Logic and Discrete Mathematics – A Computer Science Perspective.** New Jersey : Prentice-Hall. 1996.

KHARE, R.; RIFKIN, A.. **The Origin of (document) Species**. Computer Networks and ISDN Systems. 30: (1-7) 389-397 Apr 1998. Disponível em <http://www.cs.caltech.edu/~adam/papers/www/origin-of-species.html> Acesso em 19 abr. 2000.

PETRIE, C.. **Robert Cailliau on The WWW Proposal: "How It Really Happened."** Disponível em <http://computer.org/internet/v2n1/cailliau.htm> Acesso em 02 set.1999.

ROMANICHEN, E.; PEREIRA, L. D.; CAMARGO, N. F.. **UML – Exemplificando A Utilização Através De Um Sistema De Gerenciamento de Documentos para um Escritório de Contabilidade.** Curitiba : UFPR, 1999.

SUN Microsystems. **Sítio da Empresa SUN**. Disponível em <http://java.sun.com> Acesso em 20 set.2000.

UNIVERSIDADE FEDERAL DO PARANÁ. Sistemas de Bibliotecas. **Normas para Apresentação de Documentos Científicos**, 2000, pt. 2, 6, 7 e 8.

APÊNDICES

APÊNDICE 1 - UML – UM TUTORIAL – APLICADO A UM SISTEMA DE GERENCIAMENTO DE DOCUMENTOS DE UM ESCRITÓRIO DE CONTABILIDADE

Conceitos básicos e exemplos de utilização desta notação são fornecidos neste trabalho apresentado na disciplina de Tópicos em Engenharia de *Software*, do curso Mestrado em Informática da UFPR, pelos autores EDILZA ROMANICHEN, LUIS DIAS PEREIRA e NEILOR FERMINO CAMARGO.

ÍNDICE

1. Introdução.

Com a crescente demanda de sistemas desenvolvidos em linguagens orientadas a objetos, surgiram diversos métodos que procuram definir e estruturar, de acordo com certas regras, a implementação de tais sistemas. Dentre os diversos autores de métodos orientados a objetos podemos citar: Booch, Rumbaugh (OMT) e Jacobson, entre outros. Como os métodos definiam, de acordo com seu autor, uma forma de modelagem, que apesar de parecida, é divergente entre cada método, tornou-se evidente a necessidade de desenvolver uma ferramenta que pudesse unificar as formas de modelagem entre os diversos métodos. É com este objetivo que surgiu a Unified Modeling Language (UML), uma linguagem de modelagem que procura, através de diagramas, definir as principais fases dentro das diversas metodologias baseadas em objetos - análise, projeto e implementação.

Nos dias atuais é muito comum verificarmos a necessidade de modelagem, tanto de um problema virtual, como por exemplo, a implementação de um sistema computacional, quanto um problema físico, que retrata uma realidade do mundo físico, ou seja, a modelagem de um negócio, empresa ou bem material, como por exemplo, quando se tem a necessidade de construir uma casa, a primeira medida adotada é a modelagem da casa como estrutura física, e verifica-se os custos de material, mão de obra e o tempo necessário a ser despendido no projeto, verificando-se a viabilidade de construção. Sendo assim quando mencionamos modelagem, não estamos nos referindo especificamente a problemas computacionais, e sim a todos os problemas que podemos modelar de maneira estruturada.

No decorrer deste trabalho procuram-se abordar, de forma clara e objetiva, as principais características adotadas pela UML, citando-se a conceituação de classe e objeto, e também, a definição de método, modelo e linguagem de modelagem. O trabalho não tem o propósito de adotar uma metodologia padrão, porém, para melhor entendimento dos recursos da citada linguagem de modelagem, será proposto um exemplo básico, e a partir deste, será mostrado como poderiam ser modeladas as fases de análise, projeto e implementação.

2. Conceitos.

Neste capítulo procura-se abordar alguns temas que servirão de base para as definições adotadas no decorrer dos demais capítulos. A proposta inicial é de mencionar as principais características entre classe e objeto, e mostrar as diferenças entre *método, modelo e linguagem de modelagem.*

2.1. Classes e Objetos.

Segundo definição, *"Um objeto é um conceito, uma abstração, algo com limites nítidos e significado para uma aplicação. Todos os objetos têm identidade e são distinguíveis entre si. Uma classe de objetos descreve um grupo de objetos com atributos, operações e semântica comuns. Um atributo é uma propriedade dos objetos de uma classe; uma operação é uma ação que pode ser aplicada aos objetos de uma classe".* (RUMBAUGH, BLAHA, PREMERLANI, EDDY e LORENSEN, 1994).

Um objeto poderia ser uma parte de algum tipo de sistema, máquina, organização ou negócio, um objeto pode ou não existir em um mundo real, é alguma coisa que faz sentido dentro de um contexto de uma aplicação, definimos um objeto como um conceito, uma abstração, algo que possui limites claros e significados em relação a um problema. Têm o objetivo de facilitar a compreensão do mundo real no contexto de uma aplicação computacional. Com o objetivo de facilitar a compreensão de um problema, ele pode ser decomposto em objetos. Todos objetos são distintos entre si, possuindo assim sua identidade própria.

Uma classe é a descrição de um tipo de objeto, todos os objetos são instancias de uma classe, a classe descreve as propriedades e os comportamentos de um tipo de objeto, ou seja, os objetos de uma classe possuem os mesmos atributos de padrões de comportamento. Fazendo uma analogia para uma linguagem de programação, um objeto é para uma classe o que uma variável é para um tipo de dado.

2.2. Métodos, Modelos e Linguagem de Modelagem.

Existem importantes diferenças entre um método e uma linguagem de modelagem. Um método procura estruturar a ação ou o pensamento de uma pessoa com relação a um determinado problema - o método procura relatar ao usuário o que deve ser feito, como deve ser feito, quando deve ser feito e porque deve ser feito (definição do propósito de uma atividade específica), características estas que diferenciam um método de uma linguagem de modelagem. Por sua vez, métodos possuem modelos, e estes são usados para descrever alguma coisa e comunicar o resultado do uso do método. Quando se constrói um modelo também se estrutura os pensamentos a respeito do propósito da modelagem, e um modelo sempre tem um propósito, pois se um modelo não possui um propósito explícito, isto causará problemas na medida em que não se saberá como e porque o modelo deve ser usado. Um modelo é expresso em uma linguagem de modelagem.

Uma linguagem de modelagem consiste das notações (símbolos usados em um modelo), e o conjunto de regras que dizem como devem ser usadas estas notações. Quando mencionamos um conjunto de regras dentro de uma linguagem, elas podem ser de três tipos: regras sintáticas, semânticas e pragmáticas.

As regras sintáticas nos dizem como os símbolos podem parecer e como eles são combinados. As regras semânticas nos dizem o que cada símbolo significa e como ele pode ser interpretado por ele mesmo ou dentro de um contexto de outros símbolos. As regras pragmáticas definem as intenções dos símbolos pelos quais o propósito de um modelo é alcançado e fica compreensível para outros.

3. Análise.

"O propósito da análise é definir e permitir o entendimento do problema e do domínio da aplicação para que se possa construir um projeto correto. Uma análise bem feita incorpora as características essenciais do problema sem introduzir aspectos da implementação que restringem prematuramente as decisões de projeto" (RUMBAUGH, BLAHA, PREMERLANI, EDDY e LORENSEN, 1994, pg. 245).

A análise procura estabelecer o domínio do problema, sem se preocupar com

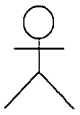
os detalhes, identificando as classes, objetos e ligações (associações) entre os objetos.

3.1. Diagrama de Casos de Uso.

Um modelo de caso de uso é descrito pela UML através de um ***diagrama de caso de uso***, e pode ser dividido em diversos diagramas. Um diagrama de caso de uso contém elementos modelados para um sistema: os atores e os casos de uso, e mostra os diferentes relacionamentos como também generalizações, associações e dependências entre estes elementos. Podemos definir dentro deste diagrama os atores, casos de uso e os limites que definem o problema (sistema).

O ***ator*** é alguém ou alguma coisa que interage com o sistema, envia ou recebe mensagem, troca informações. Uma pessoa pode ser diferentes atores em um sistema, ele possui um nome, e este nome reflete o papel do ator no sistema, o nome não deve refletir uma instância específica do ator nem a funcionalidade. Atores são ***classes*** com o estereótipo "ator" e o nome da classe reflete o papel do ator.

Um caso de uso é sempre inicializado por um ator que envia ou recebe uma mensagem ***(estímulo)***. Um ator pode receber ou enviar mensagens de um sistema ou de outro ator. Os atores podem ser também classificados como sendo ***ativos*** ou ***passivos***, o ativo inicializa o caso de uso e o passivo nunca inicia mas pode participar de um ou mais casos de uso.

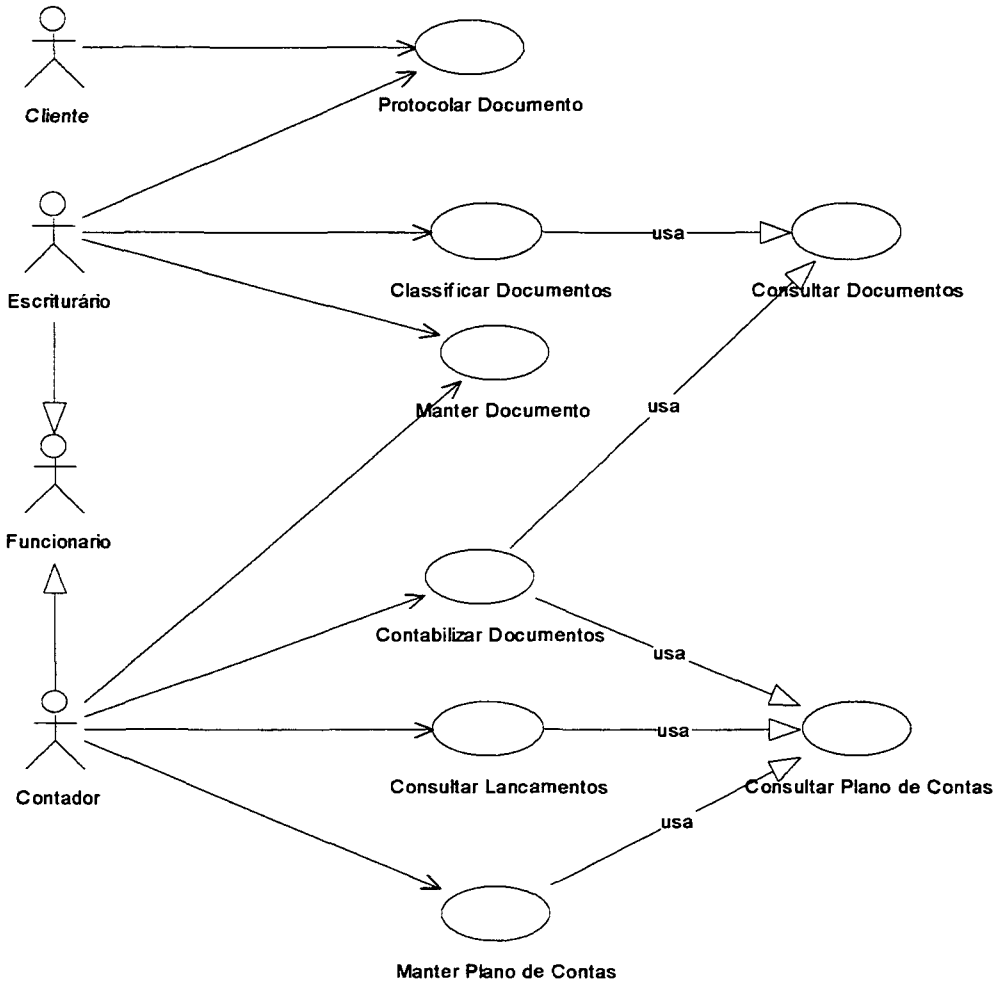


Nome do ator

Casos de Uso em UML é definido com sendo um conjunto de seqüências de ações que produz um resultado para um ator em particular. Um caso de uso é representado por uma elipse contendo o nome do caso de uso, geralmente estão contidas dentro dos limites do sistema, e podem estar conectados com um ator, associação ou uma associação de comunicação. As ações podem envolver comunicação entre uma certa quantidade de atores, como também executando

trabalhos e cálculos dentro do sistema. Um caso de uso é sempre inicializado por um ator.

Uma instância de um caso de uso é dita ***cenário*** que representa um uso atual do sistema.

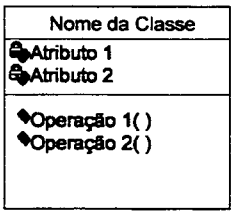


No diagrama de caso de uso mostrado anteriormente é descrito um processo de contabilização de documentos. Temos neste exemplo dois atores: ***Cliente*** é o ator principal ou ativo, que inicia o sistema entregando os documentos, e ***Funcionário*** é quem recepciona, protocola e contabiliza os documentos. O ator ***Funcionário*** possui duas generalizações: ***Escriturário e Contador.*** Temos também oito casos de usos que definem os limites do sistema (os casos de uso sempre estão dentro dos limite os atores sempre estão fora). Neste exemplo, o ***Cliente*** entrega os documentos ao ***Escriturário***, que deve iniciar o caso de uso ***Protocolar Documentos***, devolvendo o protocolo ao ***Cliente.*** O ***Escriturário*** também pode iniciar os casos de uso: ***Classificar Documentos*** ou ***Manter Documentos***, sendo que estes são generalizações de ***Consultar Documentos.*** Estes casos de uso

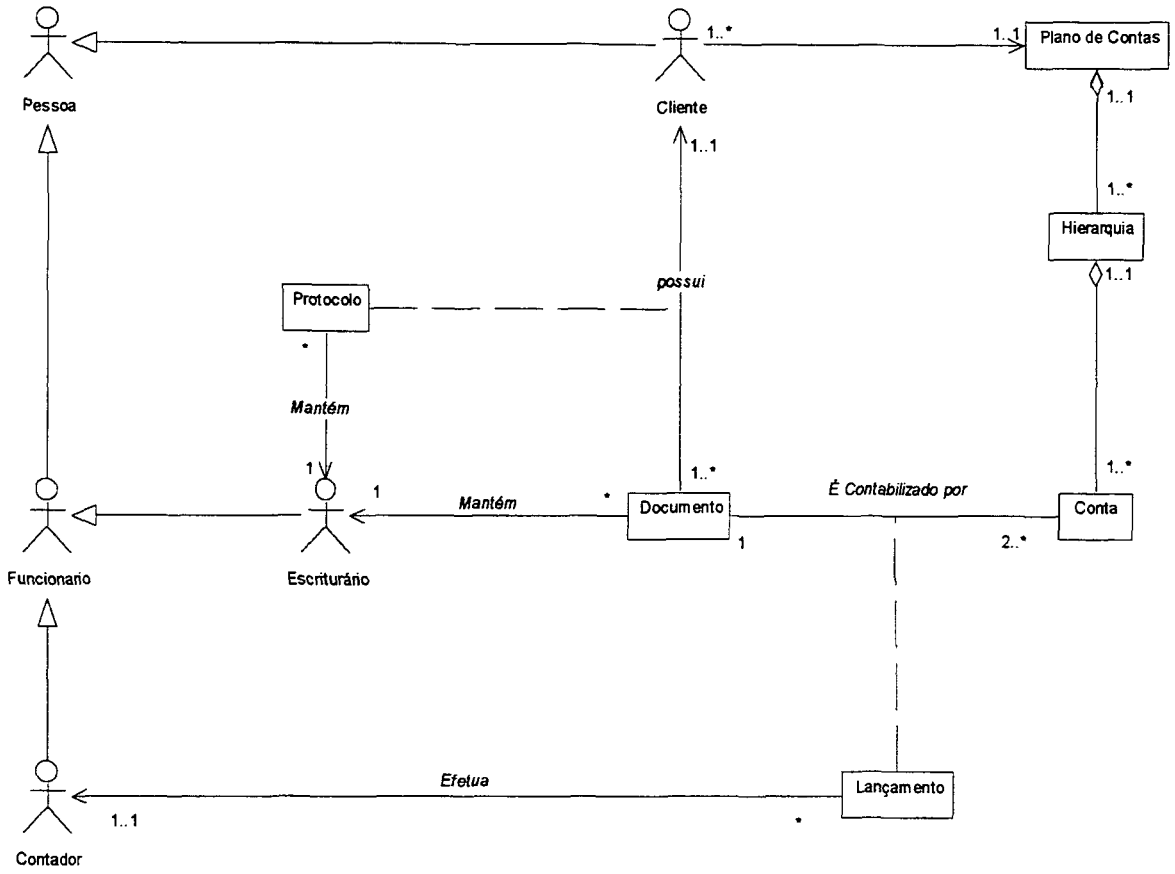
permitem que os documentos sejam disponibilizados para a Contabilização após terem sido digitalizados e classificados. Por sua vez, o ***Contador*** pode iniciar quatro casos de uso: ***Manter Documentos, Contabilizar Documentos, Consultar Lançamentos e Manter Plano de Contas,*** sendo que os últimos três citados são generalizações do caso de uso ***Consultar Plano de Contas.***

3.2. Diagrama de Classes para um Modelo Conceitual.

"Um modelo conceitual ilustra os conceitos significativos dentro do domínio do problema". (LARMAN, 199X, p: 85). O diagrama de classes resultante apresenta-se com atributos e operações ainda não completamente definidos. Uma classe possui três partes distintas: o nome que define a classe, os atributos e as operações pertencentes a esta classe. Uma classe pode estar relacionada a outra classe e pode possuir generalizações, sendo que a multiplicidade pode estar indicada nas associações. Logo abaixo mostramos a simbologia adotada para definir uma classe:



Com base no exemplo adotado, demonstra-se a modelagem de um diagrama de classes para este modelo conceitual.



No diagrama de classes mostrado anteriormente, temos sete classes e quatro generalizações, sendo elas: ***Pessoa*** (o ator também é uma classe), sendo que ***Pessoa*** possui quatro generalizações: ***Cliente e Funcionário,*** sendo que ***Funcionário*** também possui duas generalizações: ***Contador e Escriturário; Protocolo*** (Classe de Associação entre Cliente e Documento); ***Documento; Plano de Contas*** (Classe de Agregação de Hierarquia)***; Hierarquia*** (Classe de Agregação de Conta)***; Conta e Lançamento*** (Classe de Associação entre Documento e Conta). Podemos descrever o diagrama da seguinte maneira: Um ***Cliente*** pode possuir um ou mais ***Documentos*** (1..*), registrados por um ou mais ***Protocolos*** (1..*). Um ***Documento*** pertence a um único ***Cliente*** (1..1) e é registrado por um único ***Protocolo*** (1..1). Um ***Escriturário*** pode manter diversos (*) ***Documentos*** e também diversos (*) ***Protocolos***. Um ***Documento*** ou um ***Protocolo*** pode ser mantido por diversos (*) ***Escriturários***. Um ***Plano de Contas*** agrega uma ou mais (1..*) ***Hierarquias***, que por sua vez, agrega uma ou mais ***Contas*** (1..*). Um ***Documento*** é contabilizado por duas ou mais (2..*) ***Contas***, registrados através de um ou mais (2..*) ***Lançamentos***. Um ***Lançamento*** é efetuado por um único (1..1)

Contador, e um ***Contador*** pode efetuar zero ou mais (0..*) ***Lançamentos***.

O diagrama de classes para um modelo conceitual não precisaria possuir atributos ou operações, pois em uma primeira implementação poderia conter somente as classes e suas respectivas associações, sendo que poderia sofrer acréscimos de implementação durante a fase de análise, principalmente quando da modelagem do diagrama de seqüências. Durante esta modelagem, é comum surgir detalhamentos em forma de operações e atributos, que podem enriquecer o diagrama de classes.

Na finalização da fase de projeto o diagrama de classes deve conter totalmente ou quase totalmente todos os atributos e operações, para que não cause discordância com a fase de implementação.

3.2.1. Relacionamentos

Quando elementos de um modelo estão conectados entre si, nomeamos esta conexão de relacionamento. Na UML os relacionamentos podem ser de quatro tipos: Generalização/Especificação, Agregação, Associação e Dependência.

3.2.1.1. Generalização/Especificação

Relacionamento entre a superclasse (elemento geral) e a subclasse (elemento específico), onde a subclasse é herdeira da superclasse correspondente. Em nosso exemplo, ***escriturário*** e ***contador*** são generalizações de ***funcionário*** que por sua vez é generalização de ***pessoa***.

3.2.1.2. Agregação

Conexão que indica o relacionamento entre o ***Todo*** e sua ***Parte*** correspondente. Uma ***conta*** (título contábil) é parte de uma ***hierarquia*** que por sua vez é parte de um ***plano de contas*** (um plano de contas possui uma hierarquia que possui contas).

3.2.1.3. Associação

Quando um ou mais elementos possuem interdependência entre si, estando assim vinculados, dizemos estarem associados. Um ***escriturário*** que possui com conjunto de vínculos (denotado manutenção) com ***documento***, e sendo ***escriturário*** e ***documento*** classes distintas, dizemos que estes estão associados.

3.2.1.4. Dependência

Dois elementos estão relacionados por dependência quando o elemento independente muda por algum motivo, o elemento dependente também será afetado por esta mudança. Com classes a dependência existe por vários motivos, entre eles: uma classe envia mensagens para uma outra, uma classe é parte de outra classe, ou ainda, uma classe menciona outra como parâmetro para uma ou mais operações.

3.2.2. Multiplicidade

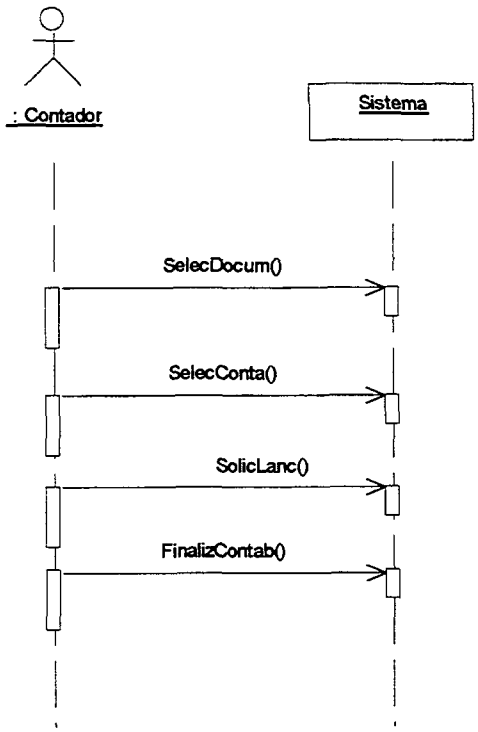
A multiplicidade indica a maneira quantitativa que muitos objetos podem participar em um relacionamento, restringindo assim a quantidade de objetos relacionados. Geralmente a multiplicidade pode ser especificada com um número ou um conjunto de intervalos. Podemos assim dizer que um ***cliente*** possui um ou muitos (1..*) documentos e um ***documento*** só pode ser possuído por um (1..1) único cliente.

3.3. Diagrama de Seqüências

Um diagrama de seqüências mostra para um cenário em particular, os eventos externos gerados por um ator, seus pedidos e os eventos internos ao sistema. Na fase de análise este diagrama trata o sistema como uma caixa preta, não importando ainda os detalhes do sistema, que somente serão conhecidos na fase de projeto.

Para o nosso já citado exemplo, demonstraremos abaixo, o diagrama de

seqüências modelado durante a fase de análise, para o caso de uso ***Contabilizar Documentos***, notemos que o sistema é tratado como um bloco único, independente das funções internas a este, e os eventos gerados pelo ator ***Contador*** são definidos na seqüência em que são gerados. Caso o ***Contador*** queira selecionar algum filtro, ele pode escolher entre um Cliente, um Protocolo, uma Data de Documento, um Estado de Documento, ou um Código de Documento específico, ou ainda, não selecionar filtro algum, através da operação ***SelecDocum( )*** uma seqüência de ações são executas pelo sistema para que o documento seja selecionado. Após selecionar o documento a ser contabilizado, inicia o processo de lançamento selecionando a conta através da operação ***SelecConta( )***. Ao final dos lançamentos relativos ao Documento, o ***Contador*** pode solicitar a operação ***FinalizContab( )***, que efetua o fechamento contábil do documento, atualizando o estado do ***documento*** para 'Contabilizado' e o estado dos ***lançamentos*** associados para 'fechado'.



3.4. Contrato.

É a descrição textual das funções (eventos) executadas pelo ator (usuário), descritas no diagrama de seqüências, são as operações de um determinado sistema descritos em detalhes, servindo como ferramenta de transportes de informações para a fase de projeto. O contrato apesar de ser bastante útil não faz parte da definição padrão da UML, sendo uma ferramenta incluída por alguns autores para servir de intercâmbio entre a fase de análise e projeto. O contrato deve possuir basicamente a seguinte estrutura: ***Nome***, ***Responsabilidade***, ***Tipo***, ***Referência cruzada***, ***Notas***, ***Exceções***, ***Saídas***, ***Pré-condições*** e ***Pós-condições***.

Exemplificando, tomaremos a operação ***SelecDocum( )***, analogamente esta definição é válida para as demais operações definidas no diagrama de seqüências.

Contrato para SelecDocum ( ):

Nome	***SelecDocum*** (codcliente : long integer, estdocum : integer, coddocum : documento, numprot : integer, dtdocum : data).
Responsabilidades	Mostrar a Tela de Consulta Documentos de acordo com os filtros solicitados, apresentando uma lista de Documentos e opções de escolha.
Tipo	Sistema.
Referências cruzadas:	
Notas	Caso de uso Contabilizar Documentos.
Exceções	Verificar se codcliente, estdocum, coddocum, numprot e dtdocum são válidos, caso contrário indicar a existência de dados errados.
Saídas	Janela de Apresentação de Documentos
Pré-condições	codcliente, estdocum, coddocum, numprot e dtdocum são conhecidos pelo sistema ou são nulos.
Pós-condições	Opção escolhida é válida – um documento (e também os atributos associados a ele) ou tecla Cancelar-Voltar.

4. Projeto.

Segundo Pressman [...pg.561]: *" A metodologia OOA/OOD (Object Oriented Analysis/Object Oriented Desing) constitui-se de uma abordagem de três passos que requer que o projetista declare o problema, defina uma estratégia de solução informal e formalize a estratégia ao identificar objetos e operações, especificar interfaces e oferecer detalhes de implementação para abstrações procedimentais e de dados. O papel do OOD é pegar as classes e objetos básicos definidos como parte da OOA e refiná-los com detalhes de projeto adicionais. Os projetos são representados usando-se uma dentre uma série de notações gráficas e uma linguagem de projeto de programa.".*

Citando também Rumbaugh [...pg. 261]: *"Durante a análise, o enfoque está sobre* ***o que*** *precisa ser feito, independentemente de como deve ser feito. No projeto, decide-se como o problema será resolvido, primeiro em alto nível e depois em níveis cada vez detalhados."*

Na fase de projeto, o resultado da análise, que não se preocupa necessariamente com detalhes, é expandido até uma solução técnica, especificando detalhadamente as operações, atributos e relacionamentos, encaminhando assim o sistema para fase de desenvolvimento.

4.1. Diagrama de Caso de Uso Real.

O diagrama de caso de uso real é a primeira atividade da fase de projeto. Sua criação é dependente da criação anterior de um caso de uso essencial. Ele descreve os atores relacionados, o propósito, fornece uma visão geral e identifica o tipo do caso de uso. Associado ao diagrama de caso de uso real está o diagrama que descreve o curso típico dos eventos, onde se identificam as ações dos atores relacionados e a resposta que o sistema deve fornecer.

Caso de Uso Real:

Caso de Uso: **Contabilizar Documentos**

Ator: Contador (Inicia)

Propósito: Gerar lançamentos contábeis

Overview: Contador consulta documentos classificados (estado classificado) e realiza o lançamento contábil de acordo com o plano de contas do cliente.

Tipo: Primário e essencial

Ref. Cruzadas:

Curso Típico dos Eventos

Ações de Atores	Resposta do Sistema
1. Início - O Contador executa função do sistema para **verificar documentos a Contabilizar**, podendo filtrar por data, protocolo, por estado do documento (classificado ou Em contabilização) ou por cliente.	
	2. **Apresenta relação de documentos** de acordo com o filtro solicitado (Caso de Uso: Consultar Documentos).
3. O Contador **escolhe o documento** a contabilizar dentre os apresentados, e **identifica suas contas** (podendo consultar) dentro do plano de contas do cliente, e gera os lançamentos.	
	4. O Sistema **mostra contas** (se solicitado), **inclui lançamento** e **atualiza o estado do documento**

para 'Em contabilização'.

5. O Contador reinicia em 1, 3 ou **efetiva a contabilização do documento**.

6. O Sistema **efetua o fechamento contábil, atualizando seu estado** para 'contabilizado' se OK, ou **mostrando mensagem** de erro se Não OK.

7. O Contador reinicia em 1, 3, 5 ou **encerra esta atividade**.

4.2. Diagrama de Interação

Os diagramas de interação demonstram as mensagens de interação entre as instâncias (e classes) dentro de um modelo de classes. O ponto inicial destas interações é o preenchimento das pós-condições das operações dos contratos.

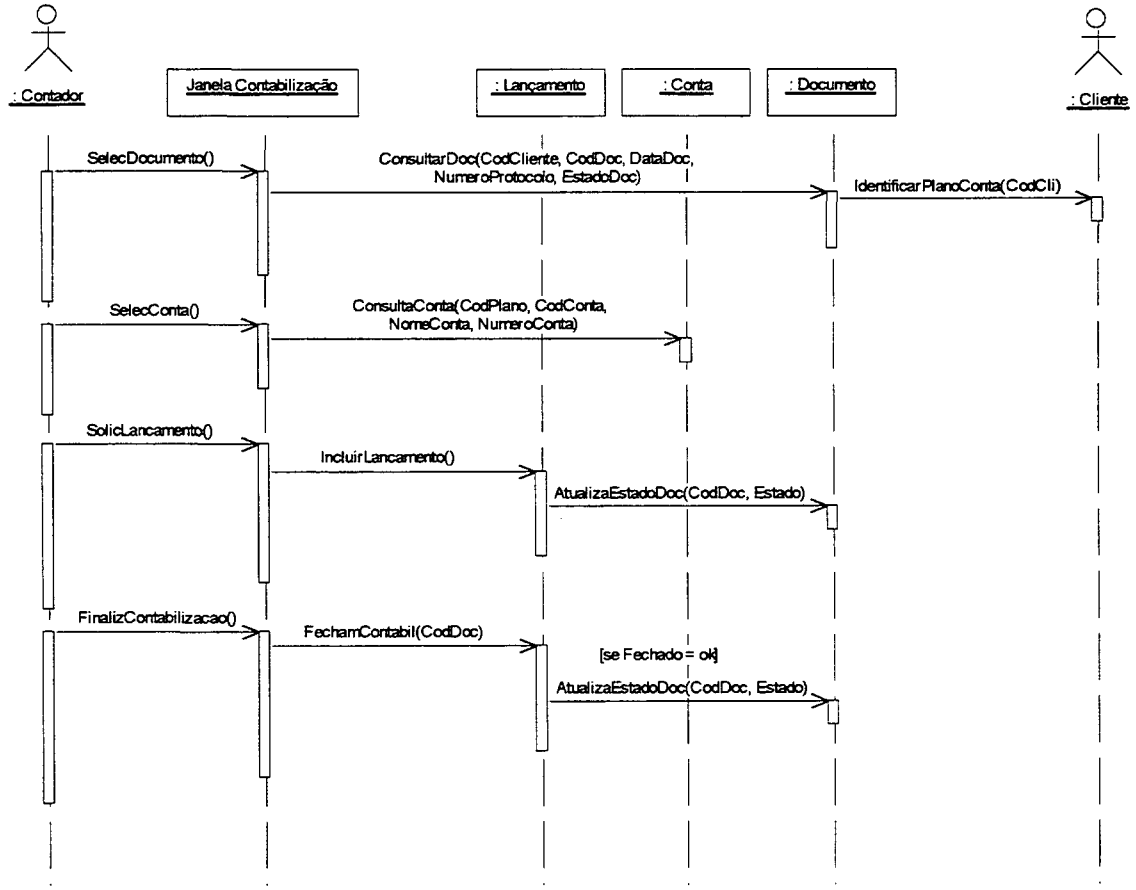
Definido por Furlan [...pg.182]: *"O diagrama de interação é utilizado para explicitar o comportamento de diversos objetos dentro de um único caso de uso, a partir das mensagens que são passadas entre eles."*.

A UML define dois tipos de diagramas de interação, sendo que cada qual pode ser utilizado para expressar mensagens idênticas ou similares, sendo eles o Diagrama de Seqüência de Eventos e o Diagrama de Colaboração.

4.2.1. Diagrama de Seqüência de Eventos.

O Diagrama de Seqüência de Eventos ilustra como os objetos interagem, com base na seqüência de mensagens, ou seja, como as mensagens são enviadas ou recebidas entre um conjunto de objetos. Este diagrama possui dois eixos: um vertical, que mostra o momento da mensagem, e outro horizontal, que apresenta uma seqüência de objetos. Também revelam a interação de determinado cenário -

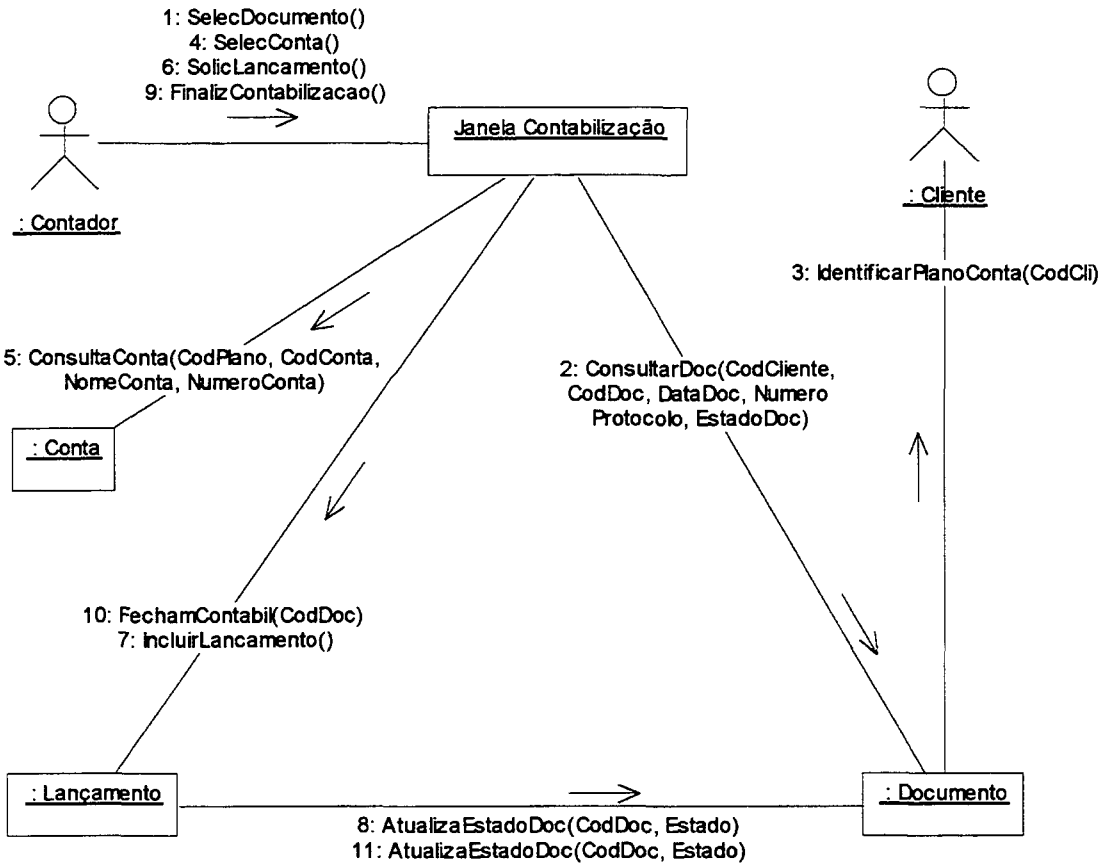
uma interação específica entre objetos que ocorre em algum ponto durante a execução do sistema, isto é, quando uma função específica é utilizada.



4.2.2. Diagrama de Colaboração

O Diagrama de Colaboração representa tanto as interações como também as ligações (uma instância de uma associação) entre um conjunto de objetos de colaboração. Este diagrama mostra os objetos atuais e suas interações no espaço, enquanto que o Diagrama de Seqüência foca o tempo.

A seqüência de tempo é indicada numerando-se as mensagens, setas indicam a orientação entre os objetos. Tal qual o Diagrama de Seqüência, o Diagrama de Colaboração ilustra uma operação, uma execução de um caso de uso, ou simplesmente um cenário de interação do sistema.



4.3. Diagrama de Classes.

Um Diagrama de Classes ilustra as especificações para interfaces e classes de software dentro de uma aplicação, sendo que durante a fase de projeto, ocorre um refinamento da fase de análise, considerando basicamente estas informações:

- classes, associações e atributos;
- interfaces, e suas operações e constantes;
- métodos;
- informações sobre os tipos dos atributos;
- navegabilidade;
- dependências.

Neste momento devemos incrementar o diagrama de classes para um modelo conceitual, definido na fase de análise. Incluindo os atributos e operações necessários. Para isto faz-se necessários especificar algumas nomenclaturas adotadas nas modelagens orientadas a objetos, e principalmente na UML.

4.3.1. Atributos

Os atributos são dados guardados pelos objetos de uma classe, são informações a respeito de uma determinada instância de um objeto. Diferentes instâncias de um objeto podem ter valores iguais ou diferentes para um dado atributo. Cada atributo possui um nome distinto dentro de uma classe. Um atributo pode ser nomeando de valor de dado, e não tem identidade, pois a ocorrência de um determinado valor por um atributo não pode ser distinguível, sendo assim um atributo não pode ser um objeto. Dentre os atributos da classe ***funcionário*** podemos citar: CodFuncionário, NomeFuncionario, Matrícula, Lotação e DataAdmissão.

4.3.2. Operações

Pode-se dizer que operação é um serviço executado por um objeto em resposta a um estímulo, é uma função ou transformação aplicada a objetos ou por estes a uma classe.

Segundo Pressman [... pg. 328]: *"Como uma primeira abordagem para derivar um conjunto de operações para os objetos de um modelo de análise, o analista pode estudar novamente a narrativa de processamento (ou declaração de escopo) do problema e selecionar aquelas operações que razoavelmente* ***pertençam*** *ao objeto. Para realizar isso, a análise gramatical é novamente estudada e verbos são isolados."*

Enquanto os substantivos, dentro da descrição de um caso de uso, são fortes candidatos a classes, os verbos são fortes candidatos a operações. Utilizando-se do curso típico dos eventos de um caso de uso, podemos facilmente identificar os substantivos e verbos correspondentes.

Todos os objetos de uma classe compartilham as mesmas operações. Cada nome de operação deve ser seguido pelos detalhes, quando estes existirem, como a lista de argumentos, que é escrita entre parênteses após o nome da operação e separados por vírgulas, e o tipo do resultado é precedido por dois pontos. O tipo de resultado, quando este existir, não deve ser omitido, visando diferenciar operações que não retornam valores.

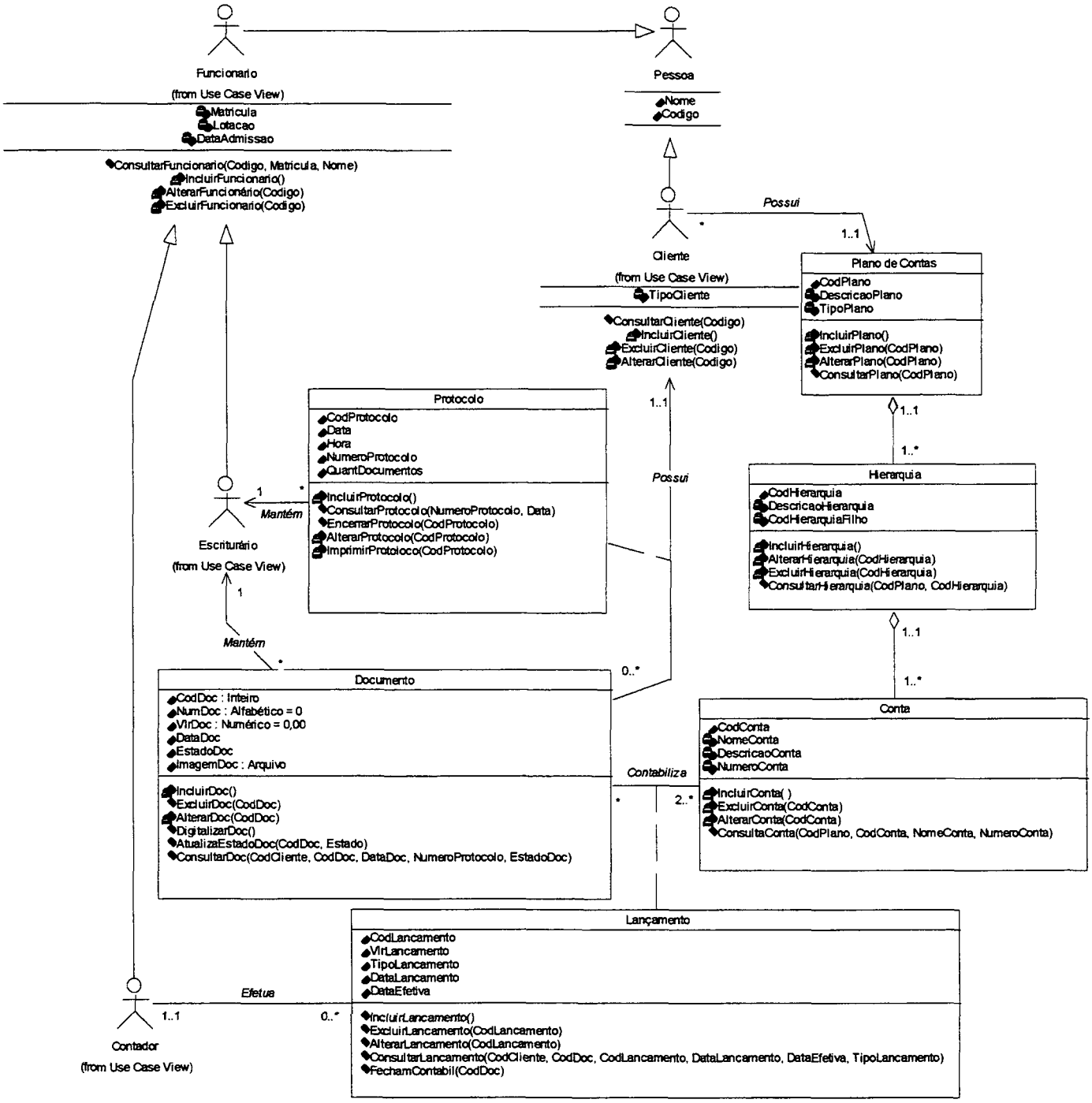
4.3.3. Método

Código implementado em linguagem de programação correspondente ao corpo da operação, ou seja, um método é uma implementação de uma operação em uma classe.

4.3.4. Mensagem

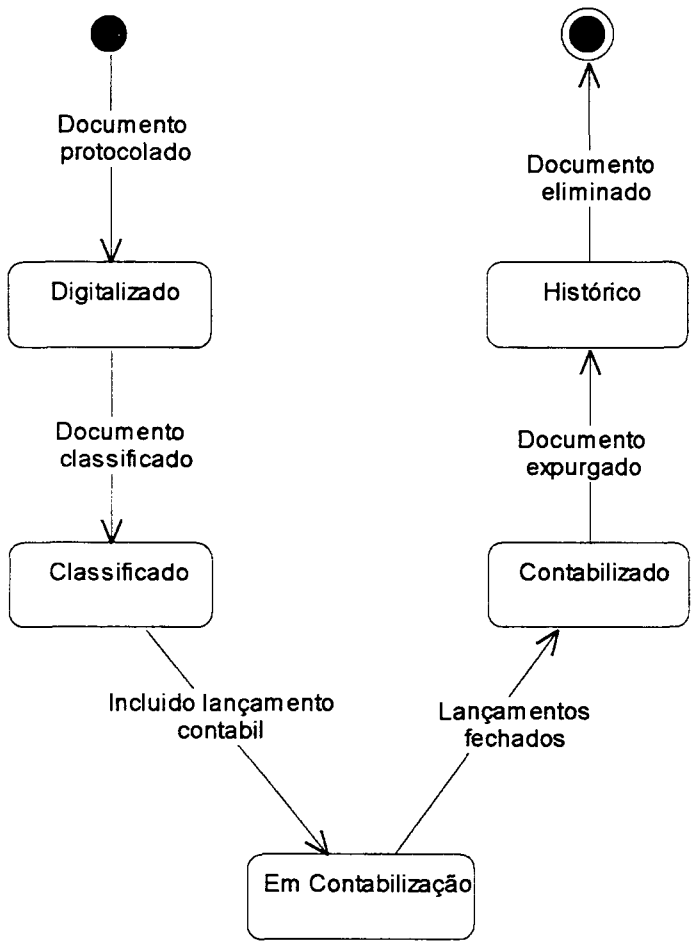
Comunicação enviada pelo objeto remetente ao objeto destinatário. Quando o objeto destinatário recebe uma mensagem (também dito evento), este pode retornar uma resposta ao objeto remetente.

As mensagens podem ser ***assíncronas*** ou ***síncronas***, diz-se ***assíncronas*** quando o objeto remetente envia uma mensagem e não interrompe seu processamento no aguardo de resultados, e ***síncronas*** quando o objeto remetente depende de resultados do objeto destinatário para continuar seu processamento.


Funcionario
(from Use Case View)
Matricula
Lotacao
DataAdmissao
ConsultarFuncionario(Codigo, Matricula, Nome)
IncluirFuncionario()
AlterarFuncionário(Codigo)
ExcluirFuncionario(Codigo)
Pessoa
Nome
Codigo
Possui
1..1
Cliente
(from Use Case View)
TipoCliente
ConsultarCliente(Codigo)
IncluirCliente()
ExcluirCliente(Codigo)
AlterarCliente(Codigo)
Plano de Contas
CodPlano
DescricaoPlano
TipoPlano
IncluirPlano()
ExcluirPlano(CodPlano)
AlterarPlano(CodPlano)
ConsultarPlano(CodPlano)
Protocolo
CodProtocolo
Data
Hora
NumeroProtocolo
QuantDocumentos
IncluirProtocolo()
ConsultarProtocolo(NumeroProtocolo, Data)
EncerrarProtocolo(CodProtocolo)
AlterarProtocolo(CodProtocolo)
ImprimirProtoloco(CodProtocolo)
Mantém
Escriturário
(from Use Case View)
Possui
Hierarquia
CodHierarquia
DescricaoHierarquia
CodHierarquiaFilho
IncluirHierarquia()
AlterarHierarquia(CodHierarquia)
ExcluirHierarquia(CodHierarquia)
ConsultarHierarquia(CodPlano, CodHierarquia)
Mantém
Documento
CodDoc : Inteiro
NumDoc : Alfabético = 0
VlrDoc : Numérico = 0,00
DataDoc
EstadoDoc
ImagemDoc : Arquivo
IncluirDoc()
ExcluirDoc(CodDoc)
AlterarDoc(CodDoc)
DigitalizarDoc()
AtualizaEstadoDoc(CodDoc, Estado)
ConsultarDoc(CodCliente, CodDoc, DataDoc, NumeroProtocolo, EstadoDoc)
Contabiliza
Conta
CodConta
NomeConta
DescricaoConta
NumeroConta
IncluirConta( )
ExcluirConta(CodConta)
AlterarConta(CodConta)
ConsultaConta(CodPlano, CodConta, NomeConta, NumeroConta)
Efetua
Lançamento
CodLancamento
VlrLancamento
TipoLancamento
DataLancamento
DataEfetiva
IncluirLancamento()
ExcluirLancamento(CodLancamento)
AlterarLancamento(CodLancamento)
ConsultarLancamento(CodCliente, CodDoc, CodLancamento, DataLancamento, DataEfetiva, TipoLancamento)
FechamContabil(CodDoc)
Contador
(from Use Case View)

4.4. Diagrama de Estados.

Algumas classes têm Diagramas de Estados para mostrar as diferentes transições de estados, que os objetos destas classes podem assumir ao longo dos eventos, que farão com que estes estados sejam alterados.



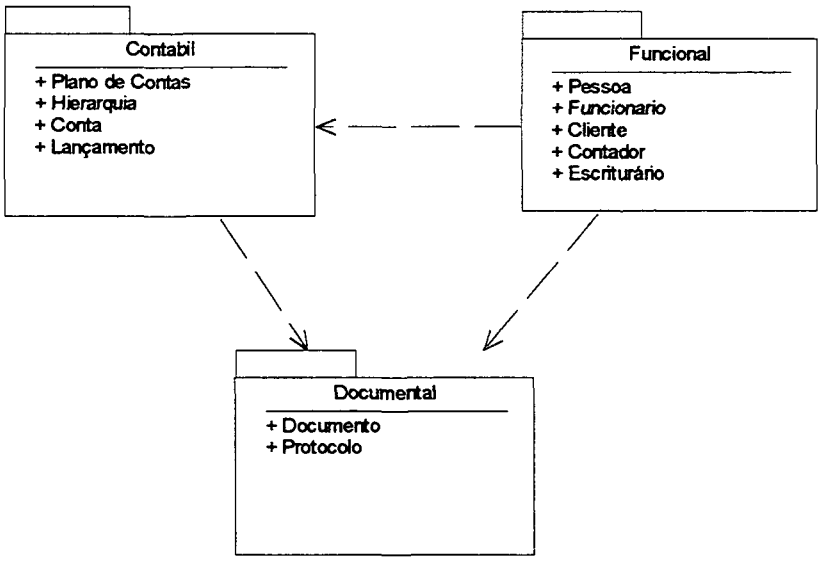
4.5. Diagrama de Pacotes.

Segundo Furlan [... pg. 83]: *"Pacote é um mecanismo de propósito geral para organizar elementos de modelo em grupos, podendo, inclusive estar aninhado dentro de outros pacotes (pacotes subordinados).".*

A arquitetura de projeto é a base de um sistema de fácil manutenção. Os pacotes podem separar a lógica da aplicação da lógica técnica. Esta separação é vital para facilitar mudanças, onde alguns segmentos podem ser alterados facilmente sem impactar os demais.

Um exemplo seria o empacotamento das Classes de Negócio, onde o projeto de suas operações é detalhado e completamente definido, e também o empacotamento das Interfaces, que possuem classes que permitem ao usuário ver os dados do sistema e entrar com novos dados. Este pacote coopera com o pacote de negócios, que contém as classes onde os dados estão atualmente armazenados, sendo que o pacote de interface chama operações do pacote de negócios para recuperar e inserir dado.

No modelo abordado até o momento, poderíamos criar o seguinte empacotamento: um pacote contendo as classes contábeis (plano de conta, hierarquia, conta e lançamento), um outro contento as classes funcionais (pessoa, funcionário, cliente, escriturário, contador) e também o pacote documental que contem as classes relacionadas diretamente ao documento.



5. Implementação.

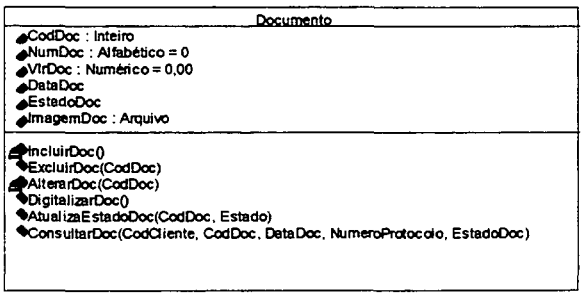
Na fase de implementação (programação ou desenvolvimento), as classes oriundas da fase de projeto são convertidas para um código produzido por uma linguagem de programação orientada a objetos.

5.1. Introdução.

Nesta parte do trabalho, procura-se introduzir uma abordagem prática,

utilizando-se como ferramenta de implementação a linguagem de programação Java.

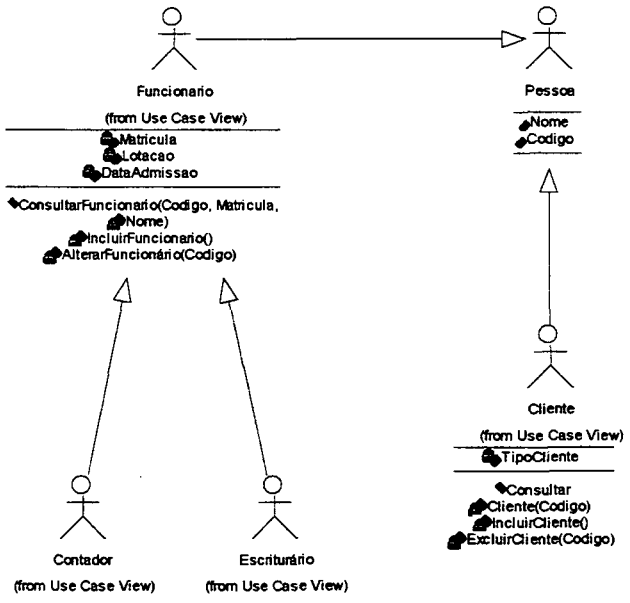
5.2. UML – Implementando uma Classe




```
/**
 * Implementação da classe Documento
 */
public class Documento {
        public long CodDoc;
        public char NumDoc = 0;
        public float VlrDoc = 0;
        public String DataDoc = "00/00/0000";
        public String EstadoDoc = "Digitalizado";
        public String ImagemDoc;

        public Documento() {
                super();
        }
        private void alterarDoc(long CodDoc) {
                /** Método para alterar dados do Documento **/
        }
        public void atualizarEstadoDoc(long CodDoc, String Estado) {
                /** Método para atualizar estado do Documento **/
        }
        public void consultarDoc(long CodCliente, long CodDoc, String DataDoc, String
NumeroProtocolo,
        String EstadoDoc) {
                /** Método para consultar dados do Documento **/
        }
```

```
        private void digitalizarDoc() {
                /** Método para digitalizar Documento **/
        }
        public void excluirDoc(long CodDoc) {
                /** Método para excluir Documento **/
        }
        private void incluirDoc() {
                /** Método para incluir Documento **/
        }

}
```

5.3. UML – Implementando Generalizações/Especializações.




```
/**
 * Implementação da classe pessoa.
 */
public abstract class Pessoa {
```

```
        public String Nome;
        public long Codigo;

        public Pessoa() {
        super();
        }

}



/**
 * Implementação da classe cliente.
 */
public class Cliente extends Pessoa {
        private int TipoCliente;


        public Cliente() {
                super();
        }
        private void alteraCliente(long Codigo) {
                /** Método para alterar cliente **/
        }
        public void consultaCliente(long Codigo) {
                /** Método para consultar cliente **/
        }
        private void excluiCliente(long Codigo) {
                /** Método para excluir cliente **/
        }
        private void incluiCliente() {
                /** Método para incluir cliente **/
        }
}


/**
 * Implementação da classe funcionario.
```

```
*/
public abstract class Funcionario extends Pessoa {
        private String Matricula;
        private String Lotacao;
        private String DataAdmissao;

        public Funcionario() {
                super();
        }
        private void alteraFuncionario(long Codigo) {
                /** Método para alterar cliente **/
        }
        public void consultaFuncionario(long Codigo, String Matricula, String Nome) {
                /** Método para consultar cliente **/
        }
        private void excluiFuncionario(long Codigo) {
                /** Método para excluir cliente **/
        }
        private void incluiFuncionario() {
                /** Método para incluir cliente **/
        }
}



/**
 * Implementação da classe escriturário.
 */
public class Escriturario extends Funcionario {

        public Escriturario() {
        super();
        }

}



/**
```

```
 * Implementação da classe contador.
 */
public class Contador extends Funcionario {

        public Contador() {
                super();
        }

}
```

6. Referências Bibliográficas.

COAD, Peter; YOURDON, Edward. ***Análise Baseada em Objetos***. Campus, 1996. ISBN 85-352-0042-8.

RUMBAUGH, James; BLAHA, Michael; PREMERLANI, William; EDDY, Frederick; LORENSEN, William. ***Modelagem e Projetos Baseados em Objetos***. 1ª Edição. Rio de Janeiro : Campus, p: 64, 1994. ISBN 85-7001-841-X.

ERIKSSON, Hans-Erik; PENKER, Magnus. ***UML Toolkit***. Wiley, 1998. ISBN 0-471-19161-2.

FOWLER, Martin; SCOTT, Kendall. ***UML Distilled – Applying the Standard Object Modeling Language***. Addison-Wesley, 1998. ISBN 0-201-32563-2.

LARMAN, Craig. ***Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design***. Prentice Hall. ISBN 0-13-748880-7.

Booch, G.; Jacobson, I.; Rumbaugh, James. ***The UML Specification Documents***. Santa Clara, CA.: Rational Software Corp, 1997.

APÊNDICE 2 - LINGUAGEM DE PROGRAMAÇÃO JAVA

Java é uma linguagem de programação orientada a objetos de propósito geral, e foi desenvolvida por James Gosling e outros parceiros na empresa *Sun Microsystems (SUN, 2001)*. Foi apresentada em maio de 1995, e adotada pela empresa *Netscape Communications* como linguagem de programação para estender as funções de seu navegador, ganhando popularidade rapidamente entre os projetistas de páginas WEB, sobretudo pela utilização de um recurso denominado "*applet*" – um pequeno programa Java que pode ser descarregado de um servidor WEB e executado em um navegador compatível do usuário.

As principais características da linguagem Java, citadas em (MARUYAMA; TANAMURA; URAMOTO, 1999, p. 25-30) (ROBERTS; HELLER; ERNEST, 2000, p. 255-264, 438-469) são:

a) portabilidade – permite a independência de plataforma. Seus programas são compilados dentro de um código executável binário ("bytecodes"), o qual posteriormente é interpretado dentro de qualquer plataforma que possua uma *Java Virtual Machine* (JVM). Este código é, em geral, mais lento que o de outras linguagens compiladas, como o C ou C++. Contudo, alternativas como compiladores *Just-In-Time* (JIT), os quais transformam o código executável binário em código de máquina nativo na hora da execução, fazem esta diferença ser minimizada;

b) segurança – não existem ponteiros em Java, que são substituídos pelo uso de vetores, objetos e outras estruturas, e existe checagem em índices de vetores para que não ultrapassem seus limites. O gerenciamento de memória é responsabilidade da JVM que possui "coleta de lixo" automática;

c) suporte embutido à entrada e saída (E/S) e Internet – está disponível para o desenvolvedor o suporte à E/S através de um conjunto de classes para tratamento de cadeias de caracteres somadas às de tratamento de arquivos. Estas classes se encontram nos pacotes "java.io" e "java.net", sendo importante destacar que todas as operações de E/S dentro e fora de uma JVM são dependentes da aprovação do gerenciador de segurança. As classes "*File*" e "*RandomAccessFile*" proporcionam funcionalidades para a navegação no sistema de arquivo local, descrição

de arquivos e diretórios, e acesso não seqüencial de arquivos. Os soquetes de E/S são suportadas pelas classes "*Socket*" e "*ServerSocket*", onde soquetes são classes abstratas, e suas instâncias fornecem recursos de E/S para uma comunicação orientada à conexão e com transferência de cadeias de dados confiável (TCP). O acesso seqüencial a arquivos é feito através das classes "*Streams*", "*Readers*" e "*Writers*". Estas classes vêem a E/S como seqüências ordenadas de *bytes*. A solução proposta utiliza as classes de tratamento de arquivos seqüenciais e soquetes de E/S;

d) suporte a caracteres internacionais (UNICODE e UTF) – utiliza dois tipos de representação de texto: UNICODE para representação interna de caracteres e cadeias de caracteres, e UTF para transferência ou gravação de texto para um arquivo ou para uma rede. Como as entidades XML utilizam como padrão de gravação o formato UTF, não foi necessário especificar a declaração de codificação de caracteres no prólogo do documento;

e) servlets – o conceito "*servlet*" é um mecanismo para invocar um programa Java reconhecido e processado no servidor WEB. Este programa pode ser chamado a partir de uma requisição HTTP e gerar resultados a serem devolvidos pela conexão HTTP solicitante. O *servlet* se apresenta como alternativa à especificação *Common Gateway Interface* (CGI), pois quando um programa CGI é executado, um novo processo é iniciado, o que pode ocasionar sobrecarga de processamento. Para solucionar este problema da especificação CGI, existem *Application Program Interface* (APIs) para CGI que podem ser chamadas do servidor WEB, como por exemplo, *Internet Server API* (ISAPI) e *Netscape Server API* (NSAPI), fazendo com que o novo processo execute no mesmo espaço de processo do servidor WEB. Porém, caso o programa CGI apresente erros de ponteiros ou endereçamento de memória, isto poderia afetar o Servidor WEB e os demais processos que compartilham o mesmo espaço de processo. Em contrapartida, o *servlet* roda em uma JVM que reside no mesmo espaço de processo do servidor WEB, porém, como o Java não

possui ponteiros e não permite o acesso em índices fora dos limites de *arrays*, não existe o risco de afetar os demais processos, apresentando-se como mecanismo de aplicações WEB eficiente e seguro. Este recurso está sendo suportado nas versões recentes de programas navegadores e servidores WEB.

APÊNDICE 3 - REPRODUÇÃO DO EXPERIMENTO

O experimento consiste de duas partes. Uma parte é um programa na linguagem Java, do tipo *servlet,* que processa no servidor WEB. A outra parte consiste de um programa na linguagem Java, que deve ser instalado e processado no cliente WEB.

A seguir, os passos para reproduzir a parte do experimento no servidor WEB, que devem ser executados uma única vez:

a) instalar o servidor *JavaWebServer 2.0 e a* biblioteca de classes JDK 1.1.8;

b) copiar o arquivo "ServletXML.class" para o diretório de classes definido pelo servidor WEB;

c) configurar o programa ServletXML para ser carregado automaticamente para a memória durante a inicialização do servidor.

Os passos para reproduzir a parte do experimento no cliente WEB são:

a) instalar a biblioteca de classes JDK 1.1.8;

b) copiar o programa "Soquete.class" para o diretório de execução;

c) do diretório de execução, solicitar a execução da seguinte linha de comando em uma janela do prompt do DOS:

java Soquete.class arquivo.txt, servidor, porta, página,

onde arquivo.txt é o nome do arquivo a ser criado com o objeto serializado persistido; servidor é o nome ou o endereço do servidor WEB; porta é um número válido para o Servidor WEB, neste caso é 8080, e a página é o processo a ser executado no servidor WEB, que para este experimento é "ServletXML".